PATENT ABSTRACTS OF JAPAN

(11)Publication number : 11-327919
(43)Date of publication of application : 30.11.1999

(51)Int.Cl. G06F 9/46
G06F 9/46

(21)Application number : 11-083237
(22)Date of filing : 26.03.1999

(71)Applicant : SUN MICROSYST INC
(72)Inventor : BOPARDIKAR SUNIL K
SAULPAUGH THOMAS
SLAUGHTER GREGORY K
ZHENG XIAOYAN

(30)Priority
Priority number : 98 47938 Priority date : 26.03.1998 Priority country : US

(54) METHOD AND DEVICE FOR OBJECT-ORIENTED INTERRUPTION SYSTEM

(57)Abstract:
PROBLEM TO BE SOLVED: To make an operating system usable together with many CPUs of different kind by processing an interrupt by a handler which is called by an interrupt dispatcher and corresponds to an identified device.
SOLUTION: When an interrupt is initiated, a microkernel generates an interrupt vector for identifying the device having generated the interrupt and supplies the interrupt vector to the interrupt dispatcher. The interrupt dispatcher refers to a device name space of a system data base and obtains an entry corresponding to the device identified with the interrupt vector. The interrupt dispatcher obtains an interrupt source entry(ISE) which cross-refers to the identified device and obtains an interrupt handler for processing the interrupt from the identified device which is specified with the ISE. The interrupt dispatcher than executes the found interrupt handler.

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（A）　(11)特許出願公開番号

特開平11－327919

(43)公開日　平成11年(1999)11月30日

(51)Int.Cl.[6]	識別記号	FI	
G06F 9/46	311	G06F 9/46	311G
	312		312

審査請求　未請求　請求項の数4　OL　（全 58 頁）

(21)出願番号　特願平11－83237

(22)出願日　平成11年(1999)３月26日

(31)優先権主張番号　09／047938

(32)優先日　1998年３月26日

(33)優先権主張国　米国（US）

(71)出願人　591064003
サン・マイクロシステムズ・インコーポレーテッド
SUN MICROSYSTEMS, INCORPORATED
アメリカ合衆国　94303　カリフォルニア州・パロ　アルト・サン　アントニオ　ロード・901

(72)発明者　サニル・ケイ・ポパーディカー
アメリカ合衆国・96087・カリフォルニア州・サニーベイル・チェリーウッド　ドライブ・537

(74)代理人　弁理士　山川　政樹

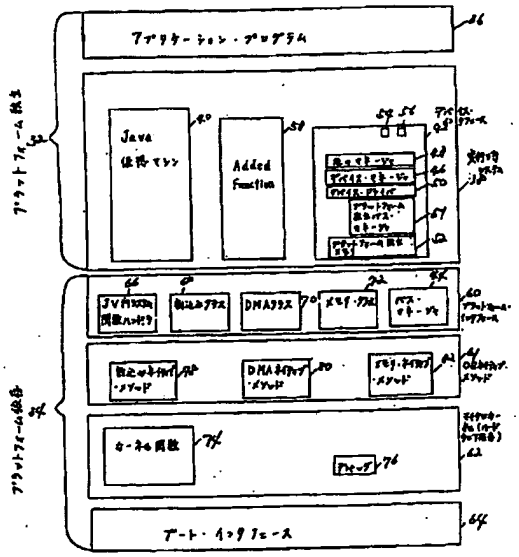
最終頁に続く

(54)【発明の名称】　オブジェクト指向割込みシステム用の方法およびデバイス

(57)【要約】

【課題】　オペレーティング・システムを、多くの異なる種類のＣＰＵとともに使用することを可能とする割込みシステムを提供する。

【解決手段】　割込み源ツリー内のそれぞれのエントリが、デバイス名前空間内の対応するエントリと相互参照し、対応する割込み源のための割込みハンドラへの参照を含む。割込みが発生すると、単一の割込みディスパッチャが呼び出され、割込み源ツリーにアクセスし、対応する割込みハンドラを実行させる。



1

【特許請求の範囲】

【請求項１】　ＣＰＵと、マイクロカーネルを含むメモリと、割込みを発生させる源を構成する複数のデバイスと、それぞれが、少なくとも１つの源に関連した複数のドライバと、システム・データベースとを有するオブジェクト指向コンピュータ・システムにおける割込みを処理する方法において、

それぞれのデバイスに対するデバイス・エントリをデータベース内に作成する段階と、

割込み源の１つを表すオブジェクトをそれぞれが含む複数の割込み源ツリー・エントリを含む割込み源ツリーをデータベース内に作成する段階と、

対応する割込み源ツリー・エントリの中のそれぞれの割込み源に関連した割込み管理ソフトウェア・コンポーネントをインストールするメソッドを含む割込み登録インタフェースを実装する段階と、

それぞれのデバイス・エントリを、対応する１つの割込み源ツリー・エントリと相互参照させる段階と、

単一の割込みディスパッチャを実行させ、割込みディスパッチャにデバイスを識別させることによって、デバイスが発生させた割込みに応答する段階と、

割込みディスパッチャによって呼び出された、識別されたデバイスに対応するハンドラで割込みを処理する段階とを含むことを特徴とする方法。

【請求項２】　ＣＰＵと、割込みを発生させることができる割込み源を構成する複数のデバイスと、それぞれが、少なくとも１つの割込み源に関連した複数のドライバとを有するオブジェクト指向コンピュータ・システムにおける割込みを処理する装置において、

それぞれが割込み源に関連したドライバによって供給される複数の割込み管理ソフトウェア・コンポーネントと、

それぞれのデバイスに対するデバイス・エントリ、および、割込み源の１つを表すオブジェクトをそれぞれが含み、対応するデバイス・エントリと相互参照し、少なくとも１つの割込み管理ソフトウェア・コンポーネントへの参照をそれぞれが含む複数の割込み源ツリー・エントリを含む割込み源ツリーを有するデータベースを含むメモリと、

識別されたデバイスに対応する割込み源ツリー・エントリによって指定された対応する割込み管理ソフトウェア・コンポーネントを実行することによって、ＣＰＵによって識別された割込みに応答する単一の割込みディスパッチャとを含むことを特徴とするデバイス。

【請求項３】　ＣＰＵと、メモリと、割込みを発生させることができる割込み源を構成する複数のデバイスとを有するオブジェクト指向コンピュータ・システムにおける割込みを処理するための機能を与える命令を含むコンピュータ可読媒体において、割込みの処理が、

メモリ内にデータベースを作成する段階と、

2

それぞれのデバイスに対するデバイス・エントリをデータベース内に作成する段階と、

割込み源の１つを表すオブジェクトをそれぞれが含む複数の割込み源ツリー・エントリを含む割込み源ツリーをデータベース内に作成する段階と、

対応する割込み源ツリー・エントリの中のそれぞれの割込み源に関連した割込み管理ソフトウェア・コンポーネントをインストールするメソッドを含む割込み登録インタフェースを実装する段階と、

それぞれのデバイス・エントリを、対応する１つの割込み源ツリー・エントリと相互参照させる段階と、

単一の割込みディスパッチャを実行させ、割込みディスパッチャにデバイスを識別させることによって、デバイスが発生させた割込みに応答する段階と、

識別されたデバイスに対応するハンドラを割込みディスパッチャで呼び出すことによって割込みを処理する段階とによって実施されることを特徴とするコンピュータ可読媒体。

【請求項４】　ＣＰＵと、メモリと、割込みを発生させることができる割込み源を構成する複数のデバイスとを有するオブジェクト指向コンピュータ・システムにおける割込みを処理するための機能を与える命令を含む搬送波上のコンピュータ・データ信号において、割込みの処理が、

メモリ内にデータベースを作成する段階と、

それぞれのデバイスに対するデバイス・エントリをデータベース内に作成する段階と、

割込み源の１つを表すオブジェクトをそれぞれが含む複数の割込み源ツリー・エントリを含む割込み源ツリーをデータベース内に作成する段階と、

対応する割込み源ツリー・エントリの中のそれぞれの割込み源に関連した割込み管理ソフトウェア・コンポーネントをインストールするメソッドを含む割込み登録インタフェースを実装する段階と、

それぞれのデバイス・エントリを、対応する１つの割込み源ツリー・エントリと相互参照させる段階と、

単一の割込みディスパッチャを実行させ、割込みディスパッチャにデバイスを識別させることによって、デバイスが発生させた割込みに応答する段階と、

識別されたデバイスに対応するハンドラを割込みディスパッチャで呼び出すことによって割込みを処理する段階とによって実施されることを特徴とするコンピュータ・データ信号。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、一般には、コンピュータ・システムで割込みを発生させるためのデバイスおよび方法に関し、詳細には、オブジェクト指向ソフトウェアを使用するコンピュータ・システムに実施された割込みシステムに関する。

【0002】
【従来の技術】コンピュータ・システムは一般に、メモリに記憶された「プログラム」に含まれる一連の特定の命令を実行することで動作する。しかし、コンピュータは、ユーザがキーボードのキーを押す、プリンタが印刷データをさらに要求する信号を送るなどの「割込み」と呼ばれるランダムに発生する外部および内部イベントにも応答しなければならない。コンピュータは通常の処理を中断し、制御の流れを、記憶されているプログラムから「割込みハンドラ」ルーチンと呼ばれる別のプログラムに一時的に迂回させなければならない。
【0003】割込み処理は、「手続き型プログラミング」として知られる従来のソフトウェアを動作させるコンピュータ、および、オブジェクト指向プログラミング手法を使用して開発されたソフトウェアを使用する新しいシステムのオペレーティング・システム（「OS」）ソフトウェアによって実行される。しかし、従来技術のオブジェクト指向割込み管理システムは、OSの移植性および性能に影響を及ぼすいくつかの限界を有する。
【0004】
【発明が解決しようとする課題】すなわち、オペレーティング・システムを、多くの異なる種類のCPUとともに使用することを可能とする割込みシステムが提供されることが望ましい。さらに、オペレーティング・システムによる割込みの処理がより高速により効率よく実施されるようにすることが望ましい。
【0005】本発明のデバイスおよび方法は、移植性があり高性能な割込み管理システムを提供することによって、現在あるこれらの限界の全てを克服することを課題とする。
【0006】
【課題を解決するための手段】本発明の特徴および利点を以下に説明する。これらには、この説明から明白なもの、あるいは本発明を実施することによって理解されるものがある。本発明の目的およびその他の利点は、本明細書の説明、請求項、および添付図面に具体的に指摘した方法、デバイスおよび製品から理解され、達成されよう。
【0007】これらおよびその他の利点を達成するため、具体例を挙げ、広く本発明を説明すると以下の通りである。本発明は、CPU、マイクロカーネルを含むメモリ、割込みを発生させる割込み源となる複数のデバイス、それぞれが1つの割込み源に関連した複数のドライバ、およびシステム・データベースを有するオブジェクト指向コンピュータ・システムで割込みを処理する方法を提供する。この方法は、それぞれのデバイスに対するデバイス・エントリをデータベース内に作成する段階、割込み源の1つを表すオブジェクトをそれぞれが含む複数の割込み源ツリー・エントリを含む割込み源ツリーをデータベース内に作成する段階、および、対応する割込み源ツリー・エントリの中のそれぞれの割込み源に関連した割込み管理ソフトウェア・コンポーネントをインストールおよび削除するメソッドを含む割込み登録インタフェースを実施する段階を含む。この方法はさらに、それぞれのデバイス・エントリを、対応する1つの割込み源ツリー・エントリと相互参照させる段階、単一の割込みディスパッチャを実行させ、割込みディスパッチャにデバイスを識別させることによって、デバイスが発生させた割込みに応答する段階、および割込みディスパッチャによって呼び出された、識別されたデバイスに対応するハンドラで割込みを処理する段階を含む。
【0008】他の態様では、本発明は、CPU、割込みを発生させることができる割込み源を構成する複数のデバイス、およびそれぞれが少なくとも1つの割込み源に関連した複数のドライバを有するオブジェクト指向コンピュータ・システムにおける割込みを処理する装置を含む。この装置は、それぞれが割込み源に関連したドライバによって供給される複数の割込み管理ソフトウェア・コンポーネントと、それぞれのデバイスに対するデバイス・エントリを含むデータベースを有するメモリと、少なくとも1つの割込み源を表すオブジェクトをそれぞれが含み、対応するデバイス・エントリと相互参照し、少なくとも1つの割込み管理ソフトウェア・コンポーネントへの参照をそれぞれが含む複数の割込み源ツリー・エントリを含む割込み源ツリーと、識別されたデバイスに対応する割込み源ツリー・エントリによって指定された対応する割込み管理ソフトウェア・コンポーネントを実行することによって、CPUによって識別された割込みに応答する単一の割込みディスパッチャとを含む。
【0009】
【発明の実施の形態】以上の概要および以下の詳細説明はともに、例示および説明のためのものであり、請求の範囲に記載した発明をさらに説明することを目的としていることを理解されたい。
【0010】添付図面は、本発明の理解を助けるためのものである。これらは、本明細書に組み込まれ、その一部を構成し、本発明の一実施形態を示し、前記説明とともに本発明の原理を説明する役割を果たす。
【0011】本発明は、オブジェクト指向オペレーティング・システム中に含まれる。開示の実施形態は、サン・マイクロシステムズ社（Sun Microsystems, Inc.）が提供するJavaプログラミング環境中で実施される。しかし本発明がこれに限定されるわけではなく、当業者なら分かるように、本発明を、その他のコンピュータ・システムに組み込むこともできる。Sun、Sun Microsystems、Sunのロゴ、Java、およびJavaをベースにした商標は、米国およびその他の国のサン・マイクロシステムズ社の商標または登録商標である。
【0012】次に、図面に示した本発明に基づく実施態

様について詳細に説明する。添付図面および以下の説明では可能なかぎり、同一または同様の要素には同じ参照番号を使用する。

【００１３】図１に本発明の使用に適したコンピュータ・システム１０を示す。コンピュータ・システム１０は、例えばＳｕｎ　ＳＰＡＲＣ、Ｍｏｔｏｒｏｌａ　Ｐｏｗｅｒ　ＰＣ、またはＩｎｔｅｌ　Ｐｅｎｔｉｍプロセッサとすることができる中央処理デバイス（ＣＰＵ）１２を含む。コンピュータ・システム１０は、広範な演算デバイスを代表する。例えばシステム１０を、家庭および事務所で広く使われている標準的なパーソナル・コンピュータとしてもよい。代わりに、システム１０が、高品位テレビジョンまたは多機能セルラ電話の受信に使用するセットトップ・ボックスのようなはるかに専門化された「スマート」システムを含んでもよい。

【００１４】ＣＰＵ１２はメモリ１４に接続される。メモリ１４は、ランダム・アクセス・メモリ（ＲＡＭ）、リード・オンリー・メモリ（ＲＯＭ）などのさまざまな種類のメモリを含むことができる。ＣＰＵ１２はさらに拡張バス１６に接続される。拡張バス１６を例えばＰＣＩバスとすることができる。バス１６には、さまざまな種類の入出力デバイス１９、２０および２２が接続される。入力デバイス１８を例えば、システム１０を電話線またはローカル・エリア・ネットワークに接続するモデムまたはネットワーク・インターフェース・カードとすることができる。入力デバイス２０を例えばキーボード、出力デバイス２２を例えばプリンタとすることができる。任意選択で、Ｉ／Ｏバス２６によって接続されたハードディスク・ドライブなどの大容量記憶デバイス２４をシステム１０に含めることができる。Ｉ／Ｏバス２６を例えばＳＣＳＩバスとすることができる。システム１０はさらに、ＣＰＵ１２の制御下にあって、メモリ１４とＰＣＩバス１６の間の直接データ転送を可能とするダイレクト・メモリ・アクセス（ＤＭＡ）コントローラ２３を含む。

【００１５】次に、図１のメモリ１４に記憶されたソフトウェアを示す図２を参照する。図２は、論理的な一連の層に配置されたソフトウェアを示す。上位層３０は、「プラットフォーム・インデペンデント」である。「プラットフォーム」という用語は、一般に、ＣＰＵ、物理メモリ、ならびに永続的に取り付けられたデバイスおよびバスを指す。したがって、プラットフォーム・インデペンデント層３２に含まれるソフトウェアは、変更なしに、既存の、または将来に開発される任意のＣＰＵに基づいた任意のプラットフォームとともに使用することができるような方法で書かれている。第２の層３４は、プラットフォーム・デペンデントである。したがって層３４のソフトウェアは、コンピュータ・システム１０の特定のプラットフォームに対してカスタマイズされていなければならない。

【００１６】プラットフォーム・インデペンデント層３２は、ユーザに対して、デスクトップ・パブリッシング、通話管理、データベース管理などの特定の動作を実行するアプリケーション・プログラム層３６を含む。アプリケーション層３６は、「Ｊａｖａ仮想マシン」（ＪＶＭ）として知られるコンポーネントを含む実行時システム３８とインタフェースする。ＪＶＭは、アプリケーション・プログラムによって生成されたマシン・インデペンデントなバイトコードの形態の命令を受け取り、それらを解釈するソフトウェア・コンポーネントである。ＪＶＭは、特定のプラットフォームとインタフェースして、所望の機能を実行する。開示の実施形態は、Ｊａｖａ仮想マシンを使用するが、当業者に周知のその他の種類の仮想マシンを使用することもできる。ＪＶＭの動作は、当業者にはよく知られており、例えば、Ｔｉｍ　Ｒｉｔｃｈｅｙ著、Ｎｅｗ　Ｒｉｄｅｒｓ　Ｐｕｂｌｉｓｈｉｎｇ社（米インディアナ州インディアナポリス）刊の書籍「Ｊａｖａ！」、Ｌｉｎｄｈａｍ、Ｙｅｌｌｉｎ共著、Ａｄｄｉｓｏｎ－Ｗｅｌｌｅｓｌｅｙ社刊「Ｔｈｅ　Ｊａｖａ　Ｖｉｒｔｕａｌ　Ｍａｃｈｉｎｅ　Ｓｐｅｃｉｆｉｃａｔｉｏｎ」（１９７７）などで論じられている。

【００１７】実行時層３８はさらに、バス１６、２６およびデバイス１８、２０、２２（図１）などのデバイスの動作をサポートするデバイス・インタフェース部分４２を含む。具体的にはデバイス・インタフェース４２は、デバイスマネージャ４６および各種マネージャ４８を含む。デバイス・インタフェース４２はさらに、それぞれのデバイス１８、２０および２２用に書かれたオブジェクト指向プログラムであるデバイス・ドライバ５０を含む。デバイス・ドライバ５０は、プラットフォーム・インデペンデント層３２に含まれており、そのため、書かれた後は、既存の、または将来に開発される任意のプラットフォームのデバイス１８、２０および２２をサポートするのに使用することができることに留意されたい。同様に、デバイス・インタフェース４２は、図１のＰＣＩバス１６およびＳＣＳＩバス２６などのバス用に書かれたオブジェクト指向プログラムであるプラットフォーム・インデペンデントなバス・マネージャ５１を含む。デバイス・インタフェース４２はメモリ・クラス５２を含む。

【００１８】デバイス・インタフェース４２はさらに、コンピュータ・システムの動作をサポートするシステム・ローダ５４およびシステム・データベースを含む。システム・データベース５６は、クライアント・ソフトウェアが、コンピュータ・システム１０の構成情報を記憶したり、検索したりすることを可能とする。具体的には、システム・データベース５６は、存在するデバイスの構成情報、インストールされているシステム・ソフトウェア・サービスについての情報、選択されているユー

ザおよびグループ属性についての情報、および必要なアプリケーション特定情報を含む。記載の実施形態では、このシステム・データベースを、Ｊａｖａシステム・データベース（ＪＳＤ）と呼ぶ。ＪＳＤの追加の詳細は、付属書Ａに記載されている。

【００１９】実行時システム３８は、入出力、ネットワーク操作、グラフィックス、印刷、マルチメディアなどの操作をサポートする追加機能５８を含む。

【００２０】プラットフォーム・デペンデント層３４は、プラットフォーム・インタフェース６０、ＯＳネイティブ層６１、マイクロカーネル６２、およびブート・インタフェース６４を含む。プラットフォーム・インタフェース６０は、仮想マシン４０から受け取ったシステム機能呼出しをサポートする仮想マシン・システム機能ライブラリ・ハンドラ６６を含む。これを、Ｊａｖａプログラム言語で書くことができる。プラットフォーム・インタフェース６０はさらに、やはりＪａｖａプログラミング言語で書かれ、コンピュータ・システム１０およびバス・マネージャ４４の割込み動作をサポートする割込みクラス６８を含む。プラットフォーム・インタフェース６０は最後に、ともにＪａｖａプログラミング言語で書かれたダイレクト・メモリ・アクセス（DMA）クラス７０およびメモリ・クラス７２を含む。

【００２１】マイクロカーネル６２は、ＣＰＵ１２特定の言語（「ネイティブ」言語）で書かれ、資源割付け、割込みプロセス、セキュリティなどのＣＰＵ１２の基本的な低水準ハードウェア機能をサポートするソフトウェア・コンポーネントから成る。具体的には、マイクロカーネル６２は、スレッド管理、例外、タイミング、物理メモリ管理、ハードウェア割込み処理、プラットフォーム制御、プロセス管理、ライブラリ管理、Ｉ／Ｏサポート、およびモニタ機能を含む複数のカーネル機能７４を含む。これらの機能を、サン・マイクロシステムズ社から市販されているＣｈｏｒｕｓマイクロカーネルによって実行してもよい。マイクロカーネル６２はさらに、デバッグ機能７６、割込みネイティブ・メソッド７８、ＤＭＡネイティブ・メソッド８０、およびメモリ・ネイティブ・メソッド８２を含む。

【００２２】プラットフォーム・デペンデント層３４の最後のコンポーネントは、ブート・インタフェース６４である。ブート・インタフェース６４は、コンピュータ・システム１０に最初に電源が投入されたときに、メモリ１４（図１）にソフトウェアをロードし、これを初期化する。ブート・インタフェース６４は、例えばフロッピー・ディスクや大容量記憶２４（図１）に記憶されたソフトウェア、または入力デバイス１８を介してネットワークから受け取ったソフトウェアをロードすることができる。

【００２３】次に、割込み処理のためのメソッドの概要を説明する。コンピュータ・システム１０用のソフトウェアをロードすると、ブート・インタフェース６４およびバス・マネージャ４４がシステム・データベースを構成する。システム・データベースは、デバイス名前空間および割込み名前空間を含む。デバイス名前空間は、ブート・インタフェース６４およびバス・マネージャ４４によって作成され、コンピュータ・システム１０のＣＰＵ、それぞれのバス、およびそれぞれのデバイスに対するエントリをオブジェクトの形態で含む。割込み名前空間は、プラットフォーム・マネージャ・ソフトウェア・コンポーネント４５によって作成され、それぞれの割込み源、すなわち、割込みを発生させることができるそれぞれのデバイスまたはバスに対するエントリをオブジェクトの形態で含む。

【００２４】割込み名前空間のオブジェクトは、割込み源ツリーの形態に編成される。それぞれのオブジェクトが割込み源エントリ（ＩＳＥ）である。デバイス名前空間のそれぞれのエントリは、ＩＳＥ、すなわち割込み名前空間のエントリと相互参照する。

【００２５】それぞれのＩＳＥは、１つまたは複数の割込み処理コード・コンポーネントへの参照を含む。これらのコンポーネントには、デバイス・ドライバによって供給される割込みハンドラ、割込みイネーブラ、割込みディスエーブラ、および割込みアクノレッジャ（ａｃｋｎｏｗｌｅｄｇｅｒ）などがある。

【００２６】割込みが発生すると、マイクロカーネルが、割込みを発生させたデバイスを識別する割込みベクトルを生成し、割込みディスパッチャを実行させ、割込みディスパッチャに割込みベクトルを供給する。割込みディスパッチャは、システム・データベースのデバイス名前空間を参照し、そこで、割込みベクトルによって識別されたデバイスに対応するエントリを求める。割込みディスパッチャは次いで、識別されたデバイスと相互参照するＩＳＥを求め、そのＩＳＥによって指定された、識別されたデバイスからの割込みを処理するための割込みハンドラを求める。割込みディスパッチャは次いで、求められた割込みハンドラを実行させる。

【００２７】以下に、本発明の一部を構成し、前述のプロセスを実装する割込みクラスおよびインタフェースの概要を示す。最初の項では、割込み源（すなわちデバイス、バスまたはＣＰＵ）を抽象的に表現し、次いで、ドライバに公表する方法を説明する。２番目の項では、割り込んだデバイスに関連付けることができるコードの種類、および、割込みを処理し管理するコードを登録する方法の概要を説明する。３番目の項では、割込みハンドラを調整し同期させる方法の概要を説明する。４番目の項では、割込みに応答して割込みハンドラをディスパッチする方法の概要を説明する。

【００２８】割込み源の要約

割込み源オブジェクト

割込みは、デバイスが、デバイス・ドライバなどのソフ

トウェアにアテンションを要求する手段である。デバイスは、ＣＰＵの中に埋め込まれたり、ＣＰＵの外部バスに直接に取り付けられたり、または、ＰＣＩなどのその他の拡張バスに取り付けられたりしている場合がある。本発明は、割込み源として割込みを発生させることができる一切のデバイスを表している。

【００２９】割込み源は、その他の割込み源に関係付けられる。例えば、ＣＰＵのバス上のデバイスは、あるＣＰＵ特定レベルでＣＰＵに割り込む。ＣＰＵの割込みレベルは割込み源、すなわちデバイス自体である。ＰＣＩバス上のデバイスが割込みを発生させると、ＣＰＵ自体が割り込まれるまで、１つまたは複数のバス・ブリッジを介して信号が伝搬される。

【００３０】割込みルーティングのトポロジとデバイス・トポロジは類似しているが、常に同一であるとは限らない。デバイス・トポロジと割込みトポロジとが異なる可能性があるため、割込み源を表すのに別のオブジェクト・クラスが必要となる。

【００３１】本発明は、ＣＰＵ、それぞれのＣＰＵ割込みレベル、およびレベルを使用したバスーデバイス結合を表す割込み源オブジェクトを作成する。図３に、ＣＰＵ、それぞれのＣＰＵレベル（この例では２つ）、バス割込み源、および２つのデバイス割込み源に対する割込み源オブジェクトの概念を示す。

【００３２】割込み源オブジェクトの編成
本発明の割込みクラスは、既知の割込み源セットを管理する。可能なアクティブな割込み源オブジェクト・セットが、ＪＳＤの割込み名前空間中に階層として編成される。

【００３３】割込み源ツリー（ＩＳＴ）
割込み名前空間（以後、割込み源ツリー（またはＩＳＴ）と称する）はＪａｖａシステム・データベース（ＪＳＤ）の中に予め作成される。割込み源ツリーの中のそれぞれのＪＳＤエントリは、割込みを通じてアテンションを要求することができるデバイスを表す。ＣＰＵ（ツリー・ルート）から、ＣＰＵレベルおよびバス（親エントリ）、最終的にデバイス（葉エントリ）にいたるＩＳＴのそれぞれの段は、割込み源の細分性がだんだんと細かくなっていくことを表す。

【００３４】デバイス・ツリーとは別のツリーを使用して割込み源を表すことには利点がある。例えば、割込みルーティングが必ずデバイスーバス接続に従うとは限らない。プラットフォームには、割込み情報をルーティングし、デコードするのにソフトウェアを必要とするものや、精巧なハードウェア補助機構を用いるものがある。

【００３５】デバイス・ドライバおよびバス・マネージャを、割込みデコード論理と割込みディスパッチング論理のさまざまな組合せを全て処理するように設計することはほぼ不可能である。ＩＳＴは、移植性のあるデバイス・ドライバ・ソフトウェアおよびバス・マネージャ・ソフトウェアを、プラットフォームの割込みハードウェアから分離し、保護するバッファの働きをする。

【００３６】割込み処理の論理および流れを伝達するのに、プラットフォームごとに異なる形状のツリーが必要な場合がある。ＩＳＴを用いることによって、プラットフォーム設計者は、ドライバが壊れることを心配せずに最新の最高性能の割り込みコントローラを使用することができる。デバイス・ドライバは、ＣＰＵに関連した割り込みコントローラ（すなわちＰＩＣ）またはバス・ブリッジ・コントローラに決してアクセスしない。

【００３７】図４に、先の例の関係付けた割込み源（図３参照）がＩＳＴの中でどのように表現されるかを示す。

【００３８】割込み源エントリ（ＩＳＥ）
それぞれの割込み源は、割込み源エントリ（ＩＳＥ）と呼ばれるＪＳＤエントリを使用して表現される。ＩＳＥは、Ｊａｖａとネイティブ・コードの両方から（時には同時に）アクセスされるＪａｖａオブジェクトである。

【００３９】割込み源エントリは、ＪＳＤのＳｙｓｔｅｍＥｎｔｒｙベース・クラスをサブクラス化する。
import java.system.database. Entry;public InterruptSourceEntry extends SystemEntry implements interruptRegistration(...)

【００４０】ＩＳＥは、ドライバが供給したコードをインストールしたり、削除したりするメソッドから成る割込み登録インタフェースを実装する。

【００４１】バスを表すＩＳＥを作成するときには、関連する子ＩＳＥ（またはデバイスＩＳＥ）の最大数を指定しなければならない。子「スロット」を予め作成することには、ハードウェア割込みレベルのＩＳＥにアクセスしなければならないネイティブ・コードにとっていくつかの利点がある。

【００４２】ネイティブ・コードにとっての第１の利点は、ページ・フォールトを防ぐために、そのオブジェクトに関連した全てのメモリを簡単にロックダウンすることができることである。ほとんどのマイクロカーネルは、割込みレベルのページ・フォールトの処理をサポートすることができない。

【００４３】第２には、ＳｙｓｔｅｍＥｎｔｒｙのリンク・リストによる方法を使用する代わりに、バスＩＳＥが、デバイスＩＳＥへの参照を予め作成した配列に記憶することができることである。ＩＳＥの中の配列を使用することによって、バス・マネージャは、子デバイスに番号をつけ、その番号を配列指標として使用することができる。これは、ハードウェア割込みレベルではとくに重要である。ＪＮＩを使用して配列に指標付けすることは、リンク・リストを実行するよりもはるかに単純である。

【００４４】図５に、バスＩＳＥの子ＩＳＥ配列を示す。

【００４５】ＩＳＴの構造
ＩＳＴは、ＪａｖａＯＳがプラットフォームを初期化するときに動的に構築される。最初に、プラットフォーム・マネージャが割込み名前空間のルートを作成し、自体を、この名前空間のマネージャとしてインストールする（名前空間マネージャに関するＪＳＤ仕様を参照のこと）。プラットフォーム・マネージャは次に、単一のルートＣＰＵ割込み源エントリ（ＩＳＥ）を作成し、次いで、（このプラットフォーム上で割込み可能な）それぞれのＣＰＵ割込みレベルに対し、子ＩＳＥを作成する。
【００４６】デバイス名前空間と割込み名前空間の調整
デバイスマネージャが、デバイスとドライバ、バスとバス・マネージャをマッチングしていくうちに、ツリーが生長し、ハンドラがインストールされる。プラットフォーム・マネージャおよび活動化されたそれぞれのバス・マネージャは、割込み源の子エントリを追加してツリーを生長させる。それぞれの新しい子ＩＳＥはＪＳＤ属性を使用して、デバイス名前空間のデバイス・エントリと相互参照する。図６に、デバイス名前空間と割込み名前空間がどのように相互参照するかを示す。
【００４７】ＩＳＴトポロジおよび管理
プラットフォーム・マネージャ、それぞれのバス・マネージャ、およびそれぞれのドライバは、ＩＳＥおよびＩＳＴとある方法で対話する。
【００４８】プラットフォーム・マネージャがＣＰＵおよびＣＰＵレベルを作成する
プラットフォーム・マネージャは割込み名前空間を作成し、次いで、単一のＣＰＵオブジェクトおよび複数のＣＰＵレベル・オブジェクトを作成する。それぞれのＪａｖａＯＳプラットフォームには、それぞれのＣＰＵレベルのための組込みハンドラ（ネイティブおよびＪａｖａ）が付属する。
【００４９】バス・マネージャがＩＳＴを伸縮させる
プラットフォームのＣＰＵバス・マネージャ（すなわちプラットフォーム・マネージャ）を含むバス・マネージャは、ドライバおよびその他のバス・マネージャに代わってＩＳＴを維持する。バス・マネージャの制御下にあるそれぞれのデバイスに対して、そのデバイスの割込み源を表す少なくとも１つのエントリがＩＳＴの中に作成される。
【００５０】デバイス・ドライバがＩＳＥの中のエントリを使用する
ドライバは、割込み源オブジェクトを構成し、次いで適当なバス・マネージャに、それをＩＳＴにインストールするよう要求する。図７に、どのコードがどのＩＳＴレベルを管理するかを示す。
【００５１】割込みコード登録
この第２の項では、割込みを処理および管理するコードを登録する方法の概要を説明する。
【００５２】割込みコードの種類
４種類のコード、すなわち割込みハンドラ、割込みイネーブラ、割込みディスエーブラ、および割込みアクノレッジャを、割込み源エントリに登録することができる。
【００５３】割込みハンドラは割込みに応答して実行される。その役割は、デバイスからのアテンションの要求を満たすこと、実時間データを安全にすること、および、できるだけ早くオペレーティング・システムに制御を返すことである。本発明は、後に説明する３種類の割込みハンドラを提供する。
【００５４】割込みイネーブラはデバイスを、割込みが可能な状態、すなわちマスクされていない状態に置く。ＣＰＵ割込みイネーブラは、ＣＰＵ割り込みコントローラを共用する全ての（内部または外部）割込みを可能にする。ＣＰＵレベル・イネーブラは、ただ１つの割込みレベルのマスクを外す。バス・イネーブラまたはデバイスイネーブラは、そのバスまたはデバイスからの割込みだけのマスクを外す。
【００５５】割込みディスエーブラは、デバイスを、割込みが不可能な状態、すなわちマスクされた状態に置く。
【００５６】割込みアクノレッジャは、ＯＳおよび／またはデバイス・ドライバが割込みを処理した（すなわち割込みに応答した）ことをハードウェアに伝える。応答は、ハンドラが実際にディスパッチされる前またはされた後に実施される。
【００５７】割込みハンドラ
本発明は、３つの割込み処理レベルを認識する。それぞれのレベルは、ハンドラおよびその種類のハンドラに対する実行コンテクストを定義する。マイクロカーネルは、これらの３つの割込み処理レベルのうちの２つを監督する。第３の割込み処理レベル（Ｊａｖａセントリック（Ｊａｖａ－ｃｅｎｔｒｉｃ）レベル）は、Ｊａｖａ仮想マシンがＪａｖａスレッドを使用してサポートする。
【００５８】割込み処理レベルに関連したそれぞれの種類のハンドラは、それ自体の特別な割込み実行コンテキストで実行される。１つの割込み源は、割込みの処理に割り当てられたこれらの種類のハンドラのうち、任意の数のハンドラを一組のハンドラとして機能させることができる。
【００５９】それぞれの割込み処理レベルは、割込み源エントリを共通のデータ交換点として使用し、状態およびデータを他のレベルに伝達することができる。これらの割込み処理レベルのうちの２つは据置きである。据置き割込みレベルは、非実時間処理に対して望ましく、ネイティブ・コードによってのみ開始される。
【００６０】３種類の割込み処理とは、
・第１レベル実時間ネイティブ割込み処理
・第２レベル据置きネイティブ割込み処理
・第３レベル据置きＪａｖａ割込み処理

である。

【0061】第1レベル・ハンドラ

第1の種類の割込みハンドラは、CPU割込みレベルで実行されるネイティブ割込みハンドラである。このハンドラはネイティブ・コードからなるが、このコードは、「C」からコンパイルされたものでも、そうでなくてもよい。アセンブリ言語からなる場合であってもこのハンドラは、そのプロセッサに対して定義された「C」呼出し規則に従う。

【0062】第1のレベル・ハンドラに入ったとき、マイクロカーネルはすでにプロセッサ・レジスタを保管しており、別の割込みスタックに切り換わっている可能性がある。どのスタックで第1レベル・ハンドラを実行するかを選択するのはマイクロカーネルである。

【0063】多くのマイクロカーネルがこの目的にCPUあたり1つの割込みスタックを充てる。他のマイクロカーネルは、単にカレント・スレッドのスタックでハンドラを実行し、このハンドラの処理が完了するとスタックをアンワインドする。

【0064】この種類のハンドラのコードは、マイクロカーネルによって呼び出されたときに実行される。割込みレベル実行コンテクストは、第1レベル・ハンドラに以下のサポート・サービスを提供する。第1レベル割込みハンドラはJNI（Javaネイティブ・インタフェース）を使用して、

・割込み源オブジェクトからデータを読み取ること、およびこれにデータを書き込むこと、

・ISTを横断し、その後に、その他の第1レベル割込みハンドラをディスパッチすること、ならびに

・第2、第3、またはメイン実行レベルで待機中のスレッドが実行されるように、割込み源オブジェクトに信号を送ること

ができる。

【0065】第1レベル・ハンドラの役割は、容量が限られているバッファでデバイスからデータを読みとるなどのデバイスの即時の実時間要求を満たすことである。

【0066】デバイスの実時間要求を満たした後に、第1レベル割込みハンドラは、第2または第3レベル・ハンドラをマイクロカーネルによって待ち行列に入れることができる。第1レベル・ハンドラは、第2および第3レベルの据置きハンドラに状態および／またはデータを、このオブジェクトのデータを取得し、設定するJNIを使用する割込み源オブジェクトを介して送る選択をすることができる。

【0067】このハンドラの「C」関数のプロトタイプは、マイクロ秒で表した、ブートから現在までの時間を含む単一のlong型パラメータを有する割込み源オブジェクトのネイティブ・メソッド・プロトタイプのそれと一致する。このハンドラは、ハンドラが割込みを処理したかどうかを知らせる整数を返す。ネイティブCPUレベル割込みハンドラは以下のように定義される。

```
typedef long firstLevelHandler(void * ise, int64#t when);
```

【0068】複数の第1レベル割込みハンドラを、それぞれ異なるレベルで同時に実行させることができる。このことは、単一CPUにもSMPシステムにも当てはまる。ただしSMPシステムでは、マイクロカーネルが、それぞれの割込みレベルの実行を直列化するので、2つのCPUが同じハンドラを同時に実行しようとすることはない。

【0069】第1レベル割込みハンドラは、システム内で最も優先順位が高いコードであり、その他の割込みハンドラおよびスレッドを優先（preempt）する。したがって、第1レベル割込みハンドラが費やす時間を最小限に抑えなければならない。

【0070】第2レベル・ハンドラ

第2の種類の割込みハンドラは、ネイティブ割込みスレッドのコンテキストで実行される。このハンドラも、「C」呼出し規則に従うネイティブ・コードから成り、単一のパラメータを有するネイティブ・メソッドとして構築される。

【0071】第1レベル・ハンドラと同様に、第2レベル・ハンドラは、その裁量に任された限られた数のサポート・サービスを有する。ネイティブ・ハンドラ（第1、第2レベル）は、JNIだけを使用してISEデータを取得し、これを設定し、同じISEに関連したその他のネイティブ・メソッドを呼び出すことができる。

【0072】第2レベル割込みハンドラは、2つの状況下で実行されるよう待ち行列に入れられる。第1レベル割込みハンドラは、第2レベル・ハンドラを待ち行列に入れることができる。第1レベル・ハンドラが存在しない場合には、マイクロカーネルが割込みに応答して第2レベル・ハンドラを自動的に待ち行列に入れる。

【0073】第2レベルネイティブ割込みハンドラは以下のように定義される。

```
typedef long secondLevelHandler(void* ise, int64#t when);
```

【0074】ネイティブ割込みスレッドは、第2レベル・ハンドラの登録プロセス中にマイクロカーネルによって作成される。ネイティブ・スレッドに割り付けられるスタックの長さは少なくとも1ページである。

【0075】第2レベル割込みハンドラは、第1レベル割込みハンドラの後に実行され、第3レベル・ハンドラおよびその他のJavaまたはネイティブ・スレッドを優先することができる。

【0076】第3レベル・ハンドラ

Java割込みハンドラはJavaスレッドのコンテキストで実行され、したがって、Java言語、Java OSおよびJDKの全てのリソースを使用することができる。

【0077】Ｊａｖａ割込みハンドラを、予め作成されたＪａｖａシステム・スレッドを含む任意のＪａｖａスレッドのコンテクストで実行させることができる。

【0078】第３レベル割込みハンドラは待ち行列に入れられ、以下の状況で実行される。第１または第２レベルのハンドラが割込み源に対して存在しない場合、デバイスが割込みを発生させると、マイクロカーネルが第３レベル・ハンドラを待ち行列に入れる。

【0079】第１または第２レベルのハンドラによって＊待ち行列に入れられた場合も、第３レベル割込みハンドラは実行される。第３レベル割込みハンドラは、第１および第２レベル割込みハンドラの後に実行され、優先順位の低いその他のスレッドを優先することができる。Ｊａｖａベースの第３レベル割込みハンドラ・メソッドは、ＤｅｖｉｃｅＩｎｔｅｒｒｕｐｔＳｏｕｒｃｅクラス内に以下のように定義され、ある有用な作業をするために導出された特定のデバイス・ドライバ・クラスによってオーバライドされる。

```
public boolean handleThirdLevelInterrupt(long when)
return true;
]
```

【0080】割込みコード用インタフェース
ＪａｖａＯＳによって認識されるためには、ハンドラ、イネーブラ、ディスエーブラおよびアクノレッジャを割込み源オブジェクトに登録しなければならない。新しいエントリがＩＳＴに追加されると、そのＩＳＥはその親※の登録されたコードを継承する。後に、子特定のコードが登録される。

【0081】Ｊａｖａインタフェースは、それぞれの種類の割込み管理コードに対して定義される。

【0082】

割込みハンドラ・インタフェース

```
public interface DeviceInterruptManager extends
InterruptManagement [
        public boolean setFirstLevelIntrHandler(int
firstLevelIntrHandler);
        public boolean setSecondLevelIntrHandler(int
secondLevelIntrHandler);
        public boolean handleThirdLevelInterrupt(long when);
        ........
]
```

ＤｅｖｉｃｅＩｎｔｅｒｒｕｐｔＳｏｕｒｃｅクラスは、ＤｅｖｉｃｅＩｎｔｅｒｒｕｐｔＭａｎａｇｅｒインタフェースを実装する。このインタフェースのｓｅｔＦｉｒｓｔＬｅｖｅｌＩｎｔｒＨａｎｄｌｅｒメソッドは、第１レベル・ネイティブ割込みハンドラの「Ｃ」アドレスを記憶するための整変数の設定に使用される整数パラメータをとる。第１レベル・ハンドラを必要とするＤｅｖｉｃｅＩｎｔｅｒｒｕｐｔＳｏｕｒｃｅ子クラスはそのネイティブ・メソッドを呼び出し、第１レベル割込みハンドラのアドレスを取り出さなければならない。次にこの子クラスは、ｓｅｔＦｉｒｓｔＬｅｖｅｌＩｎｔｒＨａｎｄｌｅｒメソッドを呼び出し、この値をＩＳＥに記憶しなければならない。このＩＳＥがＩＳＴに挿入されると、ＣｐｕＬｅｖｅｌＩｎｔｅｒｒｕｐｔＳｏｕｒｃｅクラスのメソッドは、ＩＳＥに記憶された「Ｃ」関数ポインタを使用して第１レベル割込みハンドラをこのデバイスに呼び出すことができる。

【0083】第２レベル割込みハンドラは第１レベル割込みハンドラと同様に処理される。

【0084】Ｊａｖａベースの第３レベル割込みハンドラ、ｈａｎｄｌｅＴｈｉｒｄＬｅｖｅｌＩｎｔｅｒｒｕｐｔは、子クラスにオーバライドされたＤｅｖｉｃｅＩｎｔｅｒｒｕｐｔＳｏｕｒｃｅクラスに、ダミーの実装を有する。

【0085】割込み管理インタフェース

```
public interface InterruptEnabler(...)
public interface InterruptDisabler(...)
public interface InterruptAcknowledger(...)
```

【0086】割込みレベル管理
適当な時期に手当てをしないとデバイスがデータを失う可能性があるときに、第１レベル割込みハンドラが必要となる。第１レベル割込みハンドラの持続時間はマイクロ秒で測定しなければならない。

【0087】第２レベル割込みハンドラは、マルチメディア・アプリケーションで必要となるもののような拡張実時間処理を行うときに有用である。

【0088】第３レベル割込みハンドラは、マウスおよびキーボード・イベント処理などの非実時間処理を行うときに有用である。第３レベル割込みハンドラでは、仮想マシンのガベージ・コレクタと同期をとる必要から、散発的な待ち時間が生じる可能性がある。これらの待ち時間がデバイスの管理にとって受け入れがたいものである場合には第２レベル割込みハンドラを使用しなければならない。

【0089】ドライバの役割は、それぞれの割込み処理レベルをいつ使用するかを選択することである。本発明

によって、複数の割込み処理レベルを同期させるジョブが大幅に単純化される。

【００９０】割込みハンドラの同期

同期の問題は割込み処理に付随する。本発明によって、ドライバの非割込みレベル・コードを３つの割込み処理レベルの全てと同期させることができるようになる。最初に、第２レベル・ハンドラおよび第３レベル・ハンドラの同期を検討する。

【００９１】第２および第３レベル・ハンドラを実行するそれぞれのスレッドはハンドラを実行する前に、ＩＳＥに関連したＪａｖａモニタを取得する。このようにドライバは、ＩＳＥのモニタを取得することによって、第２または第３レベル・ハンドラが実行されることを防ぐことができる。割込みハンドラがすでに実行中である場合、ハンドラがモニタを解放するまで、ドライバはブロックする。モニタが解放されている場合、ドライバはモニタを取得し、ドライバがモニタを解放するまでハンドラによる次のモニタ取得の試みをブロックする。

【００９２】スレッド・コンテキスト（すなわち第１レベル・ハンドラ）内で実行されないコードを同期させるためには、ドライバが、割込み源自体を許可および禁止＊する必要がある。それぞれの割込み源オブジェクトは、この目的の割込み許可および割込み禁止メソッドを含むインタフェースを実装する。

【００９３】図８に、割込み処理におけるＪａｖａモニタの使用を示す。

【００９４】据置き割込みハンドラ待ち行列

第１レベルおよび第２レベルのネイティブ割込みハンドラは、割込み単位で、より高位の割込みレベルに作業を据え置く選択をすることができる。ドライバが作業を据え置くための単純な機構が必要である。

【００９５】低位の割込みレベルから高位の割込みレベルに作業を据え置くためには、割込みハンドラが、現在の割込み源オブジェクトをただ単に書き留める。割込み源オブジェクトを知らせることによって、仮想マシンが、割込みを待つスレッドをウェークアップさせる。

【００９６】例えば第３レベル割込みハンドラを、予め作成したＪａｖａシステム・スレッドのコンテキスト、または、他のスレッドのコンテキストで実行することができる。待機スレッドは、以下のようなループを維持する。

```
public void run[
    while(true) [
        try[
            long when = waitForNextInterrupt();
            handleThirdLevelInterrupt(when);
        ]   catch(Throwable e)[
    ]
]
```

【００９７】図９に、据置き割込み処理の概念を示す。

【００９８】割込みディスパッチング

この項では、３レベルのハンドラを使用した割込み処理を説明する。

【００９９】割込みハンドラをディスパッチするにはＩＳＴにアクセスする必要がある。プラットフォームの初期化中、ＪａｖａＯＳのＣｐｕＬｅｖｅｌＩｎｔｅｒｒｕｐｔＳｏｕｒｃｅクラスのネイティブ・メソッドがＩＳＴにアクセス可能となる。

【０１００】この割込みクラスのネイティブ部分は、ツリーのそれぞれのエントリ（Ｊａｖａオブジェクト）が、その寿命の間、Ｊａｖａヒープにロックダウンされることを保証する。ＣＰＵを表すＩＳＥへのポインタが、静的変数に記憶され、保管されて、割込みを処理するために呼び出されたネイティブ割込みディスパッチャによってアクセスされる。

【０１０１】割込みディスパッチャ

ＪａｖａＯＳ割込みディスパッチャは、第１および第２レベル割込みハンドラを実行するネイティブ・コードのコンポーネントである。割込みディスパッチャは、マイクロカーネルの上に階層化され、自体を、マイクロカーネルがエクスポートする全ての割込みベクトルのハンドラとして実際に登録する。図１０に、マイクロカーネルの上に階層化された割込みディスパッチャを示す。

【０１０２】マイクロカーネルはＩＳＴの知識を持たない。割込みディスパッチャは、全ての割込みの唯一のハンドラを有し、マイクロカーネルに自体を提示する。この設計は、基礎をなすマイクロカーネルに、ＣＰＵベクトル割込みハンドラをインストールおよび削除する単純なインタフェース以外、ほとんど何も要求しない。

【０１０３】以下の「Ｃ」インタフェースが割込みディスパッチャをサポートする。

```
void(*native#handler)(int level) ;
void set#native#intr#handler(int level,native#handler func) ;
```

【０１０４】「ｌｅｖｅｌ」パラメータはＣＰＵレベル（またはベクトル）を指定する。「ｆｕｎｃ」パラメータは、ネイティブ割込みハンドラを指定する。ヌル・ハンドラを渡すとハンドラは削除される。マイクロカーネルは、現在の割込みレベルを唯一のパラメータとしてネイティブ・ハンドラに渡す。

【０１０５】バスおよびデバイス割込み処理

それぞれのバスおよびデバイスは関連割込みハンドラを有する。バス割込みハンドラは、割込みディスパッチャが使用するハンドラ呼出し論理を実際に増大させる特殊なデコード機能を実行する。

【０１０６】バス・ハンドラの役割は、どのデバイス割込みハンドラを次に呼び出すべきかを決定することである。デバイス割込みハンドラは、バス割込みハンドラがＪＮＩを使用して呼び出す。階層化されたバスの場合には、複数のバス割込みハンドラを呼び出して割込みをデコードすることができる。最後に、デバイスの割込みを実際に処理するデバイス・ハンドラが呼び出される。

【０１０７】バスおよびデバイス割込みハンドラは、プラットフォームの要件および性能の考慮事項に応じ、割込み処理の全てのレベルに存在することができる。ただしプロセスは常に同じである。バス割込みハンドラが、特定のバス上の割込み源を求める。デバイス割込みハンドラがこの割込みを処理する。それぞれのレベルの唯一の違いは実行コンテキストの違いであり、すなわち、割込みかスレッドか、ネイティブかＪａｖａかの違いである。

【０１０８】図１１に、バス割込みハンドラおよびデバイス割込みハンドラを示す。

＊

メソッドまたはコンストラクタ

public boolean isEnabled().

【０１１４】interface InterruptHandlers extendsFirstLevelInterruptHandler, SecondL※

メソッドまたはコンストラクタ

ハンドラ・インタフェースの収集

public boolean insert(Entry child);

＊【０１０９】図１２に、InterruptSourceクラス階層を示す。

【０１１０】性能向上

本発明は、Ｊａｖａレベル・スレッドが、InterruptSourceオブジェクトを待つだけで割込みを直接に処理することを可能にする。このプログラミング・モデルによって、ハンドラを呼び出すために、特に指定されたＪａｖａレベル・システム・スレッドに切り換える必要がなくなる。

【０１１１】例えば、従来技術のネットワーキング・ドライバは、ドライバの中の「リーダ」スレッドをウェークアップする割込みハンドラをインストールする。本発明は、「リーダ」スレッドが、特別なシステム割込みスレッドの実行に依存する代わりに、割込みを直接に待つことを可能にする。図１３に、イーサネット読取り割込みの性能向上を示す。

【０１１２】以下の項で、本発明のクラスおよびインタフェースをより詳細に説明する。

【０１１３】クラスおよびインタフェースの概要

public interface InterruptSourceEntry extends Entry

機能

この割込み源は割込み可能か？

※evelInterruptHandler, ThirdLevelInterruptHandler

機能

Ｃｐｕレベルおよびバス割込み源が、これらの全てのハンドラのインプリメンテーションを宣言する。

システムがメソッドの呼出しを試みない限り、ネイティブ・コードを供給しないことはｏｋである。これが起きた場合、NativeMethodNotFoundExceptionが送出される。

この条件を防ぐために、デフォルトの第１および第２レベル「ヌル」ハンドラが供給される。

ＪＳＤの挿入メソッドをオーバライドする。

使用可能なスロットを捜して子の配列を走査する。スロットが見つからない場合、挿入は失敗する。

ＪＳＤ内にエントリを置くことをＪＳＤに要求する（ｓｕｐｅｒ．ｉｎｓｅｒｔ（ｃｈｉｌｄ）を呼び出す）。これに失敗した場合はエラーを返す。

親の子配列に子を挿入する。

その他のネイティブ・メソッドを探索するために、ＪＮＩ呼出しを含むネイティブ側の初期化を実行するためにネイティブ・メソッドｉｎｉｔＣｈｉｌｄ（）を呼出す。

public boolean isEnabled(); この割込み源は割込み可能か？

public boolean isSrcEnabled = false; この割込み源の現在の状態を含むプール

【0115】abstract class InterruptSource extends Systemエントリ implements InterruptSourceEntry

メソッドまたはコンストラクタ 機能

InterruptSource (InterruptSource parent, String name, int maxChildSources); ＩｎｔｅｒｒｕｐｔＳｏｕｒｃｅを構成する。この割込み源を、ＪＳＤの「ｐａｒｅｎｔ」の下に挿入する。
この割込み源が、将来の割込み源の親となる場合は、子スロットの最大数を供給する。
コンストラクタが、「ｍａｘＣｈｉｌｄＳｏｕｒｃｅｓ」を使用して、ＩｎｔｅｒｒｕｐｔＳｏｕｒｃｅｓの配列を割り付ける。
この子を、親の子配列に挿入する。子への配列アクセスは最適化であり、そのため、ＪＳＤカーソルを使用して、子割込み源のリストを歩く必要はない。
プラットフォーム・マネージャが、このコンストラクタを使用するＣＰＵおよびＣＰＵレベル割込み源オブジェクトを作成する。
コンストラクタが最後に実施するのは、ネイティブ側の初期化を実行するネイティブｉｎｉｔ（）メソッドの呼出しである。例えば、割込み源をヒープに留める。そのため、ネイティブ側は割込みレベルで安全に動作することができる。

InterruptSource (String name, int maxChildSources); ＪＳＤまたは親の子配列に挿入を実施しないこと以外は、上記と同じ。
バス・マネージャおよびデバイス・ドライバがこのコンストラクタを使用することが予想される。次いでバス・マネージャがＪＳＤにエントリを挿入する。

public void inheritFromParent (InterruptSource child); 親から子に、割込み管理ルーチン（イネーブラ、ディスエーブラ、およびアクノレッジャ）をコピーする。バス・マネージャは、このフィーチャを使用して、デフォルト管理ルーチンを子および孫に伝える（階層化バス・マネー

ジャ）。

public boolean insert(Entry child);	ＪＳＤの挿入メソッドをオーバライドする。 使用可能なスロットを捜して子の配列を走査する。スロットが見つからない場合、挿入は失敗する。 ＪＳＤ内にエントリを置くことをＪＳＤに要求する（ｓｕｐｅｒ．ｉｎｓｅｒｔ（ｃｈｉｌｄ）を呼び出す）。これに失敗した場合はエラーを返す。 親の子配列に子を挿入する。 その他のネイティブ・メソッドを探索するため、ＪＮＩ呼出しを含むネイティブ側の初期化を実行するためにネイティブ・メソッドＩｎｉｔＣｈｉｌｄ（）を呼出す。
public boolean isEnabled();	この割込み源は割込み可能か？
private boolean isSrcEnabled = false;	この割込み源の現在の状態を含むブール

【0116】public class CpuInt 20*rruptSource implements erruptSource extends Inte*

メソッドまたはコンストラクタ	機能
public CpuInterruptSource(int nLevels);	ＣｐｕＩｎｔｅｒｒｕｐｔＳｏｕｒｃｅオブジェクトを構成する。その名前は、デバイス・ツリーでのＣＰＵの名前と同じである。さらに、デバイス・ツリーのＣＰＵエントリを指す「デバイス」と呼ばれるｘ－ｒｅｆ属性を追加する。ｎＬｅｖｅｌｓが、「ｍａｘＣｈｉｌｄＳｏｕｒｃｅｓ」としてＩｎｔｅｒｒｕｐｔＳｏｕｒｃｅコンストラクタに渡される。
public native int getCpuInterruptLevelCount();	このＣＰＵがサポートする割込みレベル数の整数カウントを返す。プラットフォーム・マネージャによって呼び出され、結果はＣｐｕＩｎｔｅｒｒｕｐｔＳｏｕｒｃｅコンストラクタに渡される。
private native boolean initCpu();	１回限りのＣＰＵ割込み管理初期化を実行する（ＣＰＵおよびプラットフォーム・デペンデント）。コンストラクタによって呼び出される。
public native boolean enableInterrupt()	ＣＰＵ割込みを可能にする（全レベル）
public native boolean disableInterrupt()	ＣＰＵ割込みを禁止する（全レベル）
public native void acknowledgeInterrupt()	ＣＰＵ割込みに応答する。内部または外部割込みコントローラを使用することができる（プラットフォーム／ＣＰＵ特定）。

【0117】InterruptManagement *uptSource Method or Constructor
public class CpuLevelInterruptSourceextends Interr*

メソッドまたはコンストラクタ	機能
public CpuLevelInterruptSource (CpuInterruptSource cpu, int level, int maxChildrenSharingLevel);	このCPU割込みレベル#'level'を表すCpuLevelInterruptSourceオブジェクトを構成する。このエントリの名前が、"cpu.getName()+Level"+levelになる。レベルを整数としてオブジェクトに記憶し、ネイティブ・コードがアクセスする。プラットフォームが、この割込みレベルを同時に共用することができる子装置の最大数を与える。
public int getLevel();	この割込み源に関連した割込みレベルを返す。
private native boolean initCpuLevel()	CPU割込みレベルの1回限りの初期化。CPUおよびプラットフォーム特定。何もしなくてもよい。コンストラクタによって呼び出される。
private int levelNumber;	コンストラクタからの割込みレベルを含む。
public static native int maxDevicesForLevel(int level);	プラットフォーム・ネイティブ・メソッドが、この割込みレベルを同時に共用することができるデバイスの最大数を返す。結果は、プラットフォーム・マネジャによってコンストラクタへのパラメータとして使用される。
public boolean handleThirdLevelInterrupt(long when);	このレベルの第3レベル割込みハンドラ。このレベルでの割込みに応答して、システム割込みスレッドによって呼び出される。この第3レベル・ハンドラは、全ての子の第3レベル・ハンドラを呼び出す。
public native boolean handleSecondLevelInterrupt(long when);	このレベルの第2レベル割込みハンドラ。ネイティブ・システム割込みスレッドによって呼び出される。
public native boolean handleFirstLevelInterrupt(long when);	このレベルの第1レベル割込みハンドラ。
public native boolean enableInterrupt();	CPUまたはプラットフォームの割り込みコントローラのこのレベルのマスクを外すことによってCPUレベル割込みを可能にする。
public native boolean disableInterrupt();	CPUまたはプラットフォームの割り込みコントローラのこのレベルをマスクすることによってCPUレベル割込みを禁止する。

public native void acknowledgeInterrupt();	CPUレベル割込みに応答する。CPUまたは割り込みコントローラ特定の動作。

【0118】implements InterruptManagement, InterruptHandlerspublic class DeviceIn*terruptSource extends InterruptSource Method or Constructor

メソッドまたはコンストラクタ	機能
public DeviceInterruptSource (String deviceName);	命名された指定のデバイスのDeviceInterruptSourceを構成する。 名前は、ドライバ起動後にバス・マネージャから獲得される。さらに、デバイスツリーのデバイス・エントリを指す「デバイス」と呼ばれるx－ref属性を追加する。
public DeviceInterruptSource (String deviceName, int maxChildren);	BusInterruptSourceコンストラクタがこのコンストラクタを呼び出す。
public boolean handleThirdLevelInterrupt(long when);	このデバイスの第3レベル割込みハンドラ。サブクラスが、このダミー・メソッドをオーバライドする。
protected int firstLevelIntrHandler;	これが、ネイティブ第1レベル割込みハンドラ機能のアドレスに、setFirstLevelIntrHandlerメソッドを使用して設定される。
protected int secondLevelIntrHandler;	これが、ネイティブ第2レベル割込みハンドラ機能のアドレスに、setSecondLevelIntrHandlerメソッドを使用して設定される。

【0119】implements DeviceInterruptManager, Runnablepublic abstract class BusIn※terruptSource extends DeviceInterruptSource implements InterruptManagement※

メソッドまたはコンストラクタ	機能
BusInterruptSource (Entry bus, int maxSources);	指定されたバスのBusInterruptSourceを構成する。さらに、デバイス・ツリーのデバイス・エントリを指す「デバイス」と呼ばれるx－ref属性を追加する。 整数「maxSources」は、このバスが処理する子デバイス割込み源の最大数を定義する。この整数を使用し、InterruptSourcesの配列を作成する。
public abstract boolean enableInterrupt();	サブクラスに、バス特定割込みイネーブラ・メソッドを実装させる。 親のイネーブラを継承することができる。

public abstract boolean disableInterrupt();	サブクラスに、バス特定割込みディセーブラ・メソッドを実装させる。親のディスエーブラを継承することができる。
public abstract boolean acknowledgeInterrupt();	サブクラスに、バス特定割込み応答メソッドを実装させる。親のアクノレッジャを継承することができる。

【0120】public interface Fi* *rstLevelInterruptHandler

メソッドまたはコンストラクタ	機能
public boolean handleFirstLevelInterrupt(long when);	ネイティブ第1レベル割込みハンドラのプロトタイプ。ハンドラには2つのパラメータが渡される。第1のパラメータは、オブジェクトへの通常のハンドルである。第2のパラメータは、マイクロ秒で表したタイムスタンプである。

【0121】public interface Se※ ※condLevelInterruptHandler

メソッドまたはコンストラクタ	機能
public boolean handleFirstLevelInterrupt(long when);	ネイティブ第1レベル割込みハンドラのプロトタイプ。ハンドラには2つのパラメータが渡される。第1のパラメータは、オブジェクトへの通常のハンドルである。第2のパラメータは、マイクロ秒で表したタイムスタンプである。

【0122】public interface In★ ★terruptEnabler

メソッドまたはコンストラクタ	機能
public boolean enableInterrupt();	割込みがすでに許可されていれば真を返し、そうでなければ偽を返す。この割込み源からの割込みを許可する。

【0123】public interface In☆ ☆terruptDisabler

メソッドまたはコンストラクタ	機能
public boolean disableInterrupt();	割込みがすでに許可されていれば真を返し、そうでなければ偽を返す。この割込み源からの割込みを禁止する。

【0124】public interface In◆ ◆terruptAcknowledger

メソッドまたはコンストラクタ	機能
public void acknowledgeInterrupt();	割込み処理する前および／または後に割込み源に応答する。前か後かの選択は、デバイスおよびハードウェア・デペンデントである。

【0125】以上に記載したクラスは、フロッピー・ディスク、CD－ROM、光ディスクなどのコンピュータ可読媒体に記憶される。別法としてこれらは、コンピュータ・システム10のリード・オンリー・メモリに供給されるか、またはコンピュータ・データ・キャリア・ウェーブの形態でネットワークを介して供給される。

【0126】本発明の範囲または趣旨から逸脱することなくさまざまな修正および変更を、開示のプロセスおよび製品に実施することができることは、当業者にとって明白である。本発明のその他の実施形態は、当業者が、本明細書に開示した本発明の明細および実施を考慮することによって明白となろう。以上の明細および例は単に例示のためのものであり、本発明の真の範囲および趣旨は、前記の特許請求の範囲に示されている。

【0127】付属書A

概要

Ｊａｖａシステム・データベース（ＪＳＤ）は次世代のＪａｖａ構成サポートである。ＪＳＤは、オペレーティング・システム・サービス、アプリケーション、Ｊａｖａ　ＢｅａｎｓおよびＪａｖａ　Ｄｅｖｅｌｏｐｅｒ　Ｋｉｔ（ＪＤＫ）コンポーネントが、全てのＪａｖａプラットフォームに関する複雑な構成情報を記憶したり、検索したりすることを可能にする。この構成情報には、存在するデバイス、インストールされるシステム・ソフトウェア・サービス、選択されているユーザおよびグループ属性、ならびに必要なアプリケーション特定情報が記述される（図１４参照）。

【０１２８】構成および通信ハブ

クライアント側の構成および通信ハブとして、ＪＳＤは、３つの不可欠なサービスをクライアントに提供する。すなわちＪＳＤは、構成情報（およびオペレーティング・システム情報）を記憶し、検索し、公表する。検索されたデータを、要求を出したソースに送る、すなわち公表することができる。ＪＳＤは、ネットワーク・コンピューティングおよびデスクトップ・マネージメントの信頼できる基礎を提供する。構成情報を定義および維持するためのレジストリ・サービスとして、ＪＳＤは、ネットワーク・コンピュータを管理しやすいスマートな端末に変える。ＪＳＤの重要な特徴には以下のようなものがある。

・クライアント／サーバ・実装（クライアント側でのキャッシングおよびローカル実行）
・リモート管理機能（システム管理が容易）
・単純なユーザ・インタフェース
・トランザクション・ベースのアクセス
・小さなメモリ・フットプリント（２００ｋ未満）
・イベント通知
・持続性記憶（ＦＣＳに使用可能な標準インタフェース）
・ツリー構造ベースのデータベース・エントリ階層構造（クライアントでのスマートな記憶および検索）

ＪＳＤは、１００％　ｐｕｒｅ　Ｊａｖａで書かれており、現在はＪＤＫ　１．１をサポートし、将来的には標準ＡＰＩとしてＪＤＫに移植されると予想される。

【０１２９】ＪＳＤの目的

ジェフ：「日曜の晩に、ＮＣＰマニフェスト（ｍａｎｉ＊ｆｅｓｔ）からのいくつかの資料に手を加え、この「目的」の項を追加しました。目を通して、異議のある部分または古くなっている部分を書き直すか、削除して頂くようお願いします」

ＪＳＤは以下の事項によって、ネットワーク・コンピュータの基準アーキテクチャの主要な要件を満たす。

・異種のＮＣクライアントとサーバの間の相互運用性を保証する、ネットワークコンピュータの構成管理の詳細な定義
・開発者が、自分のアプリケーションを、ＮＣおよびそのデスクトップ環境に登録または統合するための軽量なインフラストラクチャおよびＡＰＩの定義
・デスクトップの持続性（ｐｅｒｓｉｓｔｅｎｃｅ）を通したユーザ・モビリティのサポート

ＪＳＤは、クライアント側インタフェースおよびサーバ側インタフェースを有する。クライアントは、レジストリのローカル・コピーを維持する。これによって、システム構成要素またはアプリケーション（デスクトップを含む）が、データベース・エントリの値を問い合わせたり、または設定したりすることができる。サーバ側は、持続性モデルをサポートし、ネットワーク・クライアントを管理する。ＪＳＤは、予め定義された名前空間構造を有し、エントリを作成、変更、削除するためのアクセス・メソッドを含む。レジストリの探索、またはレジストリへの問合せを実施することができ、レジストリの変更をリスナに通知するイベント・モデルが実装される。セッション管理機能によって状態を保管したり、復元したりすることができ、１つのシステムから他のシステムに移動するユーザのデスクトップ持続性が保証される。

【０１３０】標準システム属性

ＪＳＤは、プラットフォーム、ユーザおよびアプリケーション特定情報を単一化されたレジストリとして維持する。データベースのそれぞれのエントリは、エントリ（またはエントリに関連した情報）を記述するのに使用する属性を有することができる。ただし属性を含まなくてもよい。標準ＪＤＫ属性にはＪＳＤ以外からもアクセス可能である。ＪＤＫは、単純なシステム属性機構を使用することによってアプリケーションおよびＪＤＫクラスの総称構成サポートを供給する。以下の表に、Ｊａｖａの標準システム属性のいくつかを示す。

【０１３１】

表１－３０　システム属性

属性名	属性値
ｊａｖａ．ｖｅｒｓｉｏｎ	バージョン番号（整数）
ｊａｖａ．ｖｅｎｄｏｒ	ベンダ名（文字列）
ｊａｖａ．ｖｅｎｄｏｒ．ｕｒｌ	ベンダのホーム・ページのＵＲＬ
ｊａｖａ．ｈｏｍｅ	Ｊａｖａのインストレーション・ディレクトリ

【０１３２】（名前文字列および値オブジェクトから成る）これらの属性は、Ｊａｖａプラットフォームの全てのバージョンに対して定義される。Ｊａｖａプラットフォームの市場がますます多様化するにつれて、追加のプ

ラットフォーム構成サポートが必要と成る可能性もある。

【0133】要件および役割

ＪＳＤデータベースは、柔軟なポピュレーション・インタフェースを使用してプラットフォームの起動時に動的にポピュレートされるように設計される。ポピュレーション・インタフェースによって、図１５に示すように、ファイル、ネットワーク、およびホストＯＳからデータを得ることができる。データベース情報の持続性は、ローカルのポピュレーティング・コンポーネントまたはクライアント／サーバ・アーキテクチャが取り扱う。例えばＷｉｎｄｏｗｓ　ＮＴでは、ＮＴレジストリからこのデータベースをポピュレートすること、およびデータベースが、ＮＴの持続性キー・サポートを利用することができる。ＪＳＤは、データベースの初期ポピュレーション後にデータのコヒーレンシ（ｃｏｈｅｒｅｎｃｙ）を維持するのに使用できるイベント機構をサポートする。

【0134】汎用機構

ＪＳＤは汎用インタフェースではあるが、本書では、提案のネットワーク・コンピュータ基準アーキテクチャに関係する、名前空間およびエントリ管理の方針について説明する。コアＪＳＤ自体は、名前空間名、それらのスキーマ、ポピュレーション方法、および基礎をなす持続性機構が存在する場合にはこれを指図しない。ＪＳＤをインスタンス化するとき、開発者は、最初の名前空間名の文字列を、それぞれの名前空間を管理するクラスとともにに与えなければならない。あるＪＳＤクラスを拡張することによって、代替の持続性機構（ＬＤＡＰ、ＡＣＡＰなど）を実装することができる。Ｊａｖａ　Ｓｙｓｔｅｍ　Ｄａｔａｂａｓｅは、アプリケーションおよび（デバイス・ドライバなどの）ＪＤＫシステム・サービスに代わって構成情報を記憶、検索するが、その情報を最初に獲得する、または解釈する方法を指図しない。ＪＳＤは、単一化された構成リポジトリであり、厳密に言うと構成マネージャではない。第２章「アーキテクチャの概要」では、提案の名前空間およびそれらのスキーマの高位の概要を示し、基礎をなすＩＩＯＰベースの通信機構について触れる。ネットワーク・コンピュータ基準アーキテクチャを明らかにするこのような定義が必要なのは、さまざまなクライアント／サーバ・システムの間の相互運用性を保証するためである。しかしＪＳＤは柔軟であるので、その他の多くの目的に使用することもできる。

【0135】第２章　アーキテクチャの概要

Ｊａｖａ　Ｓｙｓｔｅｍ　Ｄａｔａｂａｓｅは、命名されたデータベース・オブジェクトの集合である。オブジェクトの命名は、名前文字列をデータベース・エントリと関連づけるプロセスと定義される。パス名は一般に、パス名コンポーネント・セパレータ文字によって区切られた名前から成り、データベース中のエントリを一意的に識別する。エントリは、プラットフォーム、ファイル、アプリケーション、ユーザまたはデバイスなどを表す働きをすることができる。エントリを挿入することによって、その存在が、その他のソフトウェア・コンポーネントに向けて公表され、公示される。エントリは、名前文字列とＪａｖａオブジェクトの値の対である属性を有することができる。それぞれの属性に対して、ＪＳＤは、その値オブジェクトへの参照を維持する。ＪＳＤ自体が、オブジェクトを記憶するのではない。例えばＤｅｖｉｃｅ属性には、メモリ・バッファ、割込みベクトルなどのデバイスリソースがリストされる。ＪＳＤは、データベース内のエントリに関連した属性の作成、読取り、または削除に使用する標準属性インタフェースを提供する。

【0136】名前空間階層

データベースは、エントリの階層として編成される。この階層は、スーパールート・エントリとして知られる単一エントリをその頂点とする。スーパールートは、Ｊａｖａの起動手順中に作成され、初期化される。その名前は「／」である。階層内の全てのエントリは、単一エントリとしての働き、および子孫エントリのツリーのルート・エントリとしての働きをする。ツリー内のそれぞれのエントリは親エントリを１つだけ有し、子エントリはいくつでも持つことができる。全てのエントリは、それらの親エントリおよび子エントリとのリンクを含む。

【0137】名前空間

データベース階層はさらに、名前空間に細分される。名前空間は、ソフトウェア、デバイスなどの同じ種類のオブジェクトを命名する、特に指定されたエントリのツリーである。名前空間は常に、スーパールートから直接に派生する。プラットフォームの起動手順中にいくつかの標準名前空間が作成される。

【0138】名前空間マネージャ

それぞれの名前空間はデフォルトの名前空間マネージャによって管理される。名前空間マネージャは、名前空間の中にエントリを実際に記憶したり、名前空間の中のエントリにアクセスする方法を制御する。名前空間マネージャは、名前空間内のエントリのセキュリティ、記憶、および所有権属性をエクスポートする標準インタフェースを実装する。エントリは、データベースに挿入されると、その親エントリのマネージャを継承する。全ての名前空間が、特定の種類の周知のオブジェクトで完全にポピュレートされる必要はない。オブジェクト・ポピュレーションの細分性は名前空間マネージャの制御下にある。

【0139】標準データベース名前空間

Ｊａｖａプラットフォームの初期化シーケンスで、そのプラットフォームの全ての実装で使用可能な６つの周知の名前空間が作成される。初期化完了後に、動的に構成されたその他の名前空間をデータベースに追加すること

もできる。名前空間マネージャは、それぞれの標準名前空間に対して提供される。追加の名前空間も名前空間マネージャを必要とし、これらは、名前空間の作成者が提供しなければならない（標準名前空間の最上位レベルを表す図16参照）。

【0140】Software

software名前空間は、デバイス・ドライバ、アプリケーション、およびユーザ構成情報などのインストールされた、および／または使用可能なシステム・サービスのリストを含む。この情報は、ブートおよびユーザ・ログイン中に1つまたは複数のサーバ・マシンからダウンロードされた階層中に示される。クライアントとサーバの対話の詳細については第6章「クライアント／サーバの概要」を参照されたい。software名前空間は持続性である。すなわちサーバが、この名前空間内の全てのエントリに対してバックアップ記憶域を提供する。software名前空間の編成は、パブリックにアクセス可能なクライアント側JSDの多くを定義する。software名前空間の最上位レベルを以下に示す（software名前空間のサブスキーマを表す図17参照）。

【0141】それぞれのソフトウェア・カテゴリの下には、エントリ名に対するJavaの固有パッケージ命名方式を使用して配置されたサブエントリがある。すなわち、com.ibm、com.sun、com.lotus、com.ncなどの一意の会社名が、会社特定情報を区別する。一意の会社エントリの下のエントリ名および編成は会社特定である。それぞれのカテゴリの下にあるサブエントリ「Java」は、そのJDK実装のバージョンに共通のエントリを表す。これらは会社特定ではない。

【0142】Application

Applicationサブエントリ階層の例を図18に示す。

【0143】それぞれのエントリに属性を割り当てて、特定のアプリケーションに対する構成情報を示すことができる。同じアプリケーションの異なるバージョンに対するプリファレンスを表すためにその他のエントリを作成することもできる。個々の会社レベルの下の編成は、ベンダ特定である。

【0144】System

図19に例を示したSystem階層は、仮想マシンおよびNCオペレーティング・システム特定情報を含む実行時操作に関する情報を含む。デバイス・ドライバなどのロード可能なコンポーネントの構成情報は、serviceサブエントリの下に維持される。

【0145】Service

システムによってロードされたサービスの構成情報はこの階層内に維持される。サービスは一般に、デバイス・ドライバ、ネットワーキング・スタック、ファイル・システムなどのシステム・コンポーネントである。図20に可能なservice階層を示す。

【0146】Public

public階層は、一般にグローバルなデータの値を定義するそれぞれのサービスまたはアプリケーションを提供するのではなしに、このような情報の周知のリポジトリを提供する。しかし、任意のサービスまたはアプリケーションを選択して、それ自体がpublic設定をオーバライドするようにしてもよい。図21に、プロキシおよびメール・サーバ情報などの共通情報を含む階層を示す。serviceおよびapplication階層のベンダ特定部分の情報を捜す所与のサービスまたはアプリケーションを必要とせずに、個々の会社が情報を、システムの残りの部分に対してパブリックにすることも可能である。

【0147】Device

device名前空間は、ローカル・プラットフォームに見られるデバイスセットを含む。device名前空間の構造は、プラットフォームに存在するI／Oバスおよびデバイス階層の全てまたはいくつかを反映する。言い換えると、バスおよびデバイスの物理的な接続がエントリのツリーとして表される。このとき、バスが親エントリとなり、デバイスが葉エントリとなる。

【0148】デバイス・エントリ

device名前空間マネージャはdevice名前空間の内容を監督する。物理デバイスまたは論理デバイスが見つかると、ドライバおよびアプリケーションのリソースを表すデバイス名エントリが作成される。RAMディスクのような仮想デバイスもデバイス名エントリとしてこの名前空間の中に表される。device名前空間の階層は、表されたデバイスの地理的位置を直接に反映する。デバイスの地理的位置には、デバイスにアクセスするのに使用されるバスも含まれる。一般には、デバイスが葉エントリ（子エントリを持たないエントリ）で、バスが親エントリである。バス・エントリは常に、デバイス・エントリの親エントリかまたはグランドペアレント・エントリである。バス・エントリを、他のバス・エントリの子エントリにすることもできる。例えば、PCIバス・ブリッジを、Sバスデバイスを含むSバスに繋げてもよい。このシナリオを、バスーデバイス・トポロジを示す図22の中央のパネルに示した。PCIバス・エントリが、Sバス・エントリの親エントリに、Sバス・エントリが、Sバスデバイス・エントリの親エントリ担っている。Device名前空間は、Javaの起動プロセス中に構成される。device名前空間マネージャ（デバイス・マネージャとも呼ばれる）は、TreePopulatorインタフェースのサービスを使用し、一度に1エントリずつdevice名前空間を構築する。TreePopulatorインタフェースは、ドライバの探索を手伝うそれぞれのプラットフォーム・

デバイスについての（属性の形態の）情報をデバイス・マネージャに提供する。

【0149】Ｉｎｔｅｒｆａｃｅ

ｉｎｔｅｒｆａｃｅ名前空間は、（ｄｅｖｉｃｅ、ｓｏｆｔｗａｒｅなどの）その他の名前空間の中のタイプによるオブジェクトのクロス・レフェレンスである。例えばデバイス・ドライバは、ｄｅｖｉｃｅ名前空間内のエントリの管理で競合する。あるドライバがそのデバイスを要求すると、クロス・レフェレンス・インタフェース・エントリが、ｉｎｔｅｒｆａｃｅ名前空間の中のｄｅｖｉｃｅと呼ばれる周知のエントリの下に挿入される。例えば、シリアル・デバイス・ドライバが（ｄｅｖｉｃｅ名前空間内の）シリアル・デバイスを要求すると、クロス・レフェレンスＳｅｒｉａｌＤｅｖｉｃｅエントリがｉｎｔｅｒｆａｃｅ名前空間に追加される。インタフェース・トポロジを示す図２３に、シリアル・マウスについてこの関係を示す。Ｊａｖａアプリケーション、Ｊａｖａアプレットおよびｊａｖａ　Ｂｅａｎｓは、ｉｎｔｅｒｆａｃｅ名前空間の中のインタフェース・タイプによってデバイスを探索し、その結果、プラットフォームの物理接続から隔離される。アプリケーションは、より単純化されたａｌｉａｓ名前空間を使用して、上図に示した／ａｌｉａｓ／ｍｏｕｓｅなどの一般的なデバイスの場所を見つけることができる。（デバイス・ドライバの詳細については、別に入手可能なＪＤＩ仕様を参照されたい。）

【0150】インタフェース・エントリ

ｉｎｔｅｒｆａｃｅ名前空間は、標準Ｊａｖａランゲージ・プログラミング・インタフェースを実装するデバイス・ドライバのセットを含む。ドライバがデバイスに割り当てられると、デバイス・マネージャは、そのドライバによってどのインタフェースが実装されたかを書き留める。次いでこれらのインタフェースは、ｉｎｔｅｒｆａｃｅ名前空間に自動的に記録される。新しいデバイスが発見されるかまたは、既存のデバイスが取り除かれると、ｉｎｔｅｒｆａｃｅ名前空間の内容が変更される。しかし、インタフェース・エントリとデバイス・エントリは１対１には対応しない。これは、１つのデバイス・ドライバが、２つ以上のプログラミング・インタフェースをサポートすることがあるからである（例えばＳｅｒｉａｌＤｅｖｉｃｅ、ＰｏｗｅｒＭａｎａｇｅｍｅｎｔなど）。ｉｎｔｅｒｆａｃｅ名前空間のトポロジはむしろフラットであり、デバイス・ドライバによって実装されたそれぞれの種類のＪａｖａインタフェースに対して１つの親エントリを含む。それぞれの親エントリは、親インタフェースを実装するデバイスを表す子エントリを＊有することができる。ただしこのような子エントリを持たなくてもよい。例えばシリアル・ドライバはシリアル・ポートを管理し、ＳｅｒｉａｌＤｅｖｉｃｅインタフェースを実装する。シリアル・ドライバによってアクティブに管理されたそれぞれのシリアル・ポートに対して、ｉｎｔｅｒｆａｃｅ名前空間内にエントリが作成される。ＳｅｒｉａｌＤｅｖｉｃｅエントリは、ＳｅｒｉａｌＤｅｖｉｃｅインタフェース・エントリの子エントリとして作成される。このクロス・レフェレンス割当てプロセスを以下のようにまとめることができる。Ｊａｖａ　Ｓｅｒｖｉｃｅ　Ｌｏａｄｅｒ（ＪＳＬ）が、適当なデバイスへのドライバ・クロス・レフェレンス・エントリでｉｎｔｅｒｆａｃｅ名前空間をポピュレートする。このクロス・レフェレンス割当てプロセスを図２４に示す。同図で、デバイス・ドライバはインタフェースを実装し　（＃１）、デバイス・マネージャは、ｄｅｖｉｃｅ名前空間内のデバイスをこのドライバに割り当てる（＃２）。

【0151】Ｃｏｎｆｉｇ

ｃｏｎｆｉｇ名前空間は、クライアント、ユーザ、およびアプリケーション構成情報を維持するのに使用される。この名前空間は、サーバのＪＳＤ実装でのみ使用される。クライアントでは、ｓｏｆｔｗａｒｅ名前空間が持続性であり、ハードウェア・プラットホームおよび現在ログインしているユーザに特定の情報を含む。サーバでは、ｃｏｎｇｆｉｇｕｒａｔｉｏｎ名前空間が持続性である。詳細については第６章「クライアント／サーバの要点」を参照のこと。

【0152】Ａｌｉａｓ

ａｌｉａｓ名前空間のエントリはｉｎｔｅｒｆａｃｅ名前空間のエントリを参照し、いくつかのシステムデバイスの単純な周知の論理的命名方式を提供する。エイリアスは、システム管理者によって明示的に定義される。この名前空間の一般的なエイリアスの例は、エントリ／ｉｎｔｅｒｆａｃｅ／ｄｅｖｉｃｅ／ｐｒｉｎｔｅｒ／ｈｐＩｎｋｊｅｔ＃１を参照することができる／ａｌｉａｓ／ｓｙｓｔｅｍ／ＤｅｆａｕｌｔＰｒｉｎｔｅｒである。

【0153】Ｔｅｍｐ

ｔｅｍｐ名前空間は一時記憶として使用可能である。ｔｅｍｐの中のエントリの変更を指示するイベントが生成されるが、エントリ自体は持続性でない。アプリケーションは、ＪＳＤの機能セットおよび、この一時記憶機能を利用することができる。

【0154】要約

名前空間の要点を以下の表に示す。

表１－３１　名前空間の要点

名前空間	ポピュレーション機構	説明
ｓｏｆｔｗａｒｅ	ブートおよびユーザ・ログイン中にサーバか	ａｐｐｌｉｃａｔｉｏｎ、ｓｙｓｔｅｍ、ｓｅｒｖｉｃｅ、および

	らポピュレートされる	ｐｕｂｌｉｃサブディビジョン（ｓｕｂｄｉｖｉｓｉｏｎ）が、持続性データをクライアント上に保持する
ｄｅｖｉｃｅ	ＴｒｅｅＰｏｐｕｌａｔｏｒインタフェースによってポピュレートされる	サブディビジョンはバス／デバイス特定である
ｉｎｔｅｒｆａｃｅ	デバイス・マネージャによってポピュレートされる	ドライバ情報を、ｄｅｖｉｃｅおよびａｌｉａｓ名前空間へのクロス・レフェレンスとともに保持する
ｃｏｎｆｉｇ	持続性記憶域からサーバ側ＪＳＤへポピュレートされる	持続性情報を記憶するときにのみサーバ側で使用される
ａｌｉａｓ	システム管理者	ｉｎｔｅｒｆａｃｅ名前空間とのクロス・レフェレンス
ｔｅｍｐ	任意のサブシステムまたはアプリケーション	一時記憶

【０１５５】第３章　システム・エントリ
ＪＳＤは、それぞれがパス名によって識別される情報の原子性単位であるエントリの階層から成る。エントリは、それに関連した１つまたは複数の属性を有することができる。属性は、エントリをさらに説明するものである。この章では、エントリの構造、エントリの状態、ならびにエントリに関連した操作およびイベントについて説明する。
エントリ構造
ＪＳＤのそれぞれのエントリは、エントリ・インタフェースを実装するベース・クラスＳｙｓｔｅｍＥｎｔｒｙによって定義される。このインタフェースは、データベースの中のエントリを操作するのに使用される一般的なメソッドのセットを定義する。それぞれのエントリは、データベース階層、管理方式およびそれぞれのエントリに関連した属性の維持に使用される以下の属性を定義する。
・名前
・状態
・親
・子
・ロック
・クッキー（不明瞭なデータ）
・マネージャ
・世代番号
・属性
【０１５６】名前
エントリ名は、ｊａｖａ．ｌａｎｇ．Ｓｔｒｉｎｇオブジェクトであり、ＵＲＬでアドレス可能な任意の文字を含むことができる。ヌル参照または空の文字列であってはならないこと、および、そのエントリが属する親の下の子エントリのいずれとも異なっていなければならないこと以外に、エントリ名に制限はない。ＳｙｓｔｅｍＥｎｔｒｙクラスは、パス名のコンポーネント・セパレータ文字であるフォワード・スラッシュ（「／」）を名前文字列中に使用できないという追加の制約事項を定義する。
【０１５７】
状態エントリは、３つの状態、すなわちドラフト（ｄｒａｆｔｅｄ）状態、公表（ｐｕｂｌｉｓｈｅｄ）状態、および削除（ｄｅｌｅｔｅｄ）状態をとることができる。エントリの状態については、後にこの章の「エントリ状態」の項で詳細に論じる。
【０１５８】親エントリ
親エントリを参照する。公表状態では、親エントリは常に有効である。
【０１５９】子エントリ
エントリは、その全ての子エントリへの参照を維持する。子への参照は、列挙法によって使用可能である（第５章「データベース・ナビゲーション」を参照されたい）。
【０１６０】ロック
干渉なしにエントリを検査したり、または変更したりすることができるようにリーダーライタ・ロック機構が用意されている（ＳｙｓｔｅｍＤａｔａｂａｓｅＬｏｃｋ）。Ｊａｖａモニタ機構（同期キーワード）は、同期キーワードの範囲内にオブジェクトを排他的にロックさせる（コード・ブロックまたはメソッド全体）。しかし、データベース・エントリ・ロックは、エントリがロックされている間に複数のメソッド呼出しが実施されることを考慮している。データベース・エントリ・ロックはさらに、複数リーダまたは排他的ライタを許容する。

リーダとライタは相互排除である。リーダはいくつあってもよいが、ライタは１つだけである。書込みロックには、Ｔｒａｎｓａｃｔｉｏｎオブジェクトが必要である。１つのトランザクションのみが、書込みロックを獲得し所有する。所有トランザクション自体が、追加のリーダおよびライタを許可するかどうかの方針を決定する。現在、Ｔｒａｎｓａｃｔｉｏｎクラスは、単一の排他的ライタのみを許可する。詳細については第４章「トランザクション」を参照されたい。

【０１６１】クッキー（Ｃｏｏｋｉｅ）

クッキーは通常、不明瞭なオブジェクト参照である。しかし本書で提案している基準アーキテクチャでは、クッキーがよりはっきりとした機能を有する。エントリ・クッキーは、そのエントリに割り当てられたサービス・クラス・オブジェクトを保持するのに使用される。クッキーの値は、ＳｙｓｔｅｍＥｎｔｒｙ　ｓｅｔＣｏｏｋｉｅ（）メソッドを使用して設定される。この値はｇｅｔＣｏｏｋｉｅ（）を介して取得される。

【０１６２】マネージャ

それぞれのエントリは、マネージャと呼ばれるロード可能なソフトウェア・コンポーネントによって管理される。それぞれの名前空間は、マネージャ・クラスを与える。エントリは、データベースに挿入されると、その親のマネージャを継承する。マネージャは、エントリの方針を設定するのに使用される。マネージャは、セキュリティ・チェックを実行したり、またはエントリの振舞いに影響を及ぼすその他の判断を下したりすることができる。さらにマネージャは、エントリに持続性を与える機構である。

【０１６３】世代番号

エントリ世代番号は、単調に増加する６４ビット数（Ｊａｖａ　ｌｏｎｇ）であり、エントリが変更されるたびにエントリのステータスとは無関係に増分される。これは、イベント処理のオーバヘッドを必要としない単純なコヒーレンシ方式のポーリング機構を提供する。エントリの変更とは、データベース中でのエントリの挿入または削除、あるいは属性の追加、変更または削除であると定義される。

【０１６４】属性

それぞれのエントリには属性が関連づけられる。ただし属性を持たなくてもよい。属性は、名前（ｊａｖａ．ｌａｎｇ．Ｓｔｒｉｎｇ）と値（ｊａｖａ．ｌａｎｇ．Ｏｂｊｅｃｔ）から成る。属性を追加、変更、削除、および検索するメソッドがＥｎｔｒｙインタフェースに提供されている。存在する全ての属性名を発見するために、属性名を列挙することができる。これ以上の構造は属性に加えられない。名前および値の実際のタイプの決定は属性のユーザに完全に委ねられている。ＪＳＤは、暗黙的ブール属性をサポートする。属性だけが追加されて値が与えられない場合、その属性はブール属性であるとみなされる。その存在は真の値を示し、その不在は偽の値を指示する。属性を追加、除去、および変更するメソッドが提供される。既存の属性に値が割り当てられると、変更が起こる。属性値（参照）ではなく値オブジェクト自体が何らかの方法で変更された場合（例えばバイト・アレイ中のバイトの変更など）、ＪＳＤは、変更が起こったことを検出することができない。同じ属性名および値をそのエントリに再び割り当てることによってその他に通知するのは、このような変更を行ったエンティティの責任である。その結果、世代番号が増分されイベントが生成される。属性名を指定する属性関連メソッドのいずれかに対して、ＥｎｔｒｙＰｒｏｐｅｒｔｙＮａｍｅＮｏｔＦｏｕｎｄＥｘｃｅｐｔｉｏｎを送出し、（読取ったり、削除したりするよう）指定された属性名が存在しないことを指示することができる。いくつかのエントリ・タイプは、何がリーガルな属性名を示すかについて制約を置くことができる。このようなエントリは、ＥｎｔｒｙＰｒｏｐｅｒｔｙＮａｍｅＩｎｖａｌｉｄＥｘｃｅｐｔｉｏｎを送出することができる。

【０１６５】属性読取り

ｇｅｔＰｒｏｐｅｒｔｙＣｏｕｎｔ（）メソッドを呼び出すことによって、そのエントリに関連した属性の数が返される。ｇｅｔＰｒｏｐｅｒｔｙＮａｍｅｓ（）メソッドは、そのエントリに関連した全ての属性名を列挙することができる列挙を返す。属性名が与えられると、その値は、ｇｅｔＰｒｏｐｅｒｔｙＶａｌｕｅ（）メソッドを介して獲得することができる。

【０１６６】属性の追加および属性値の変更

ａｄｄＰｒｏｐｅｒｔｙ（）メソッドは、新しい属性を追加するか、または既存のメソッドを修正する。追加する属性の名前が存在していない場合には、属性名が作成され、与えられた値が新しい属性名と関連づけられる。エントリが公表状態にある場合、ＥｎｔｒｙＰｒｏｐｅｒｔｙＩｎｓｅｒｔＥｖｅｎｔが関係消費者に対して生成される。新しい属性を指示するａｄｄＰｒｏｐｅｒｔｙ（）はｎｕｌｌオブジェクト参照を返す。そのエントリに対する属性名がすでに存在する場合、その新しい値がその属性名に関連づけられ、公表エントリに対して、ＥｎｔｒｙＰｒｏｐｅｒｔｙＶａｌｕｅＣｈａｎｇｅＥｖｅｎｔが生成される。この場合、ａｄｄＰｒｏｐｅｒｔｙ（）が以前の値オブジェクトへの参照を返す。

【０１６７】属性の削除

属性は、ｒｅｍｏｖｅＰｒｏｐｅｒｔｙ（）メソッドを使用してエントリから削除される。属性を削除するときには、公表エントリに対してＥｎｔｒｙＰｒｏｐｅｒｔｙＲｅｍｏｖｅＥｖｅｎｔが生成される。削除された属性値への参照が返される。

【０１６８】エイリアス・エントリ

ＳｙｓｔｅｍＡｌｉａｓＥｎｔｒｙを使用して、データベース中のその他のエントリ、またはその他のエイリア

スを参照する。エイリアスは、他のエントリへの参照を保持し、（エイリアス・エントリとは別に）それ自体の属性を維持することができる。しかし、エイリアスが子を持つことは許されていない。リンク鎖が循環してはならない。ただし参照鎖が、ツリーおよび名前空間の境界を横切ってもよい。複数のエイリアスまたはリンクが１つのエントリを指すことができる。しかしこれは、それぞれのエイリアスが参照されたエントリの親であることを意味しない。実際、エイリアスへエイリアシングされたエントリは参照を維持しない。したがって、エイリアシングされたエントリが削除されても、エイリアスは、削除されたエントリを参照することができる。このエイリアスは、削除されたエントリを参照するので、ターゲット・エントリはガベージ・コレクションの対象とはならない。このように、エイリアス自体も削除するのでなければ、エイリアスを、動的エントリの参照に使用してはならない。いくつかの非標準名前空間マネージャは、エイリアスの作成をサポートしない可能性がある。しかし標準名前空間マネージャは全て、それらをサポートする。

【０１６９】エントリ状態

エントリは、３つの異なる状態、すなわちドラフト（ｄｒａｆｔｅｄ）状態、公表（ｐｕｂｌｉｓｈｅｄ）状態、および削除（ｄｅｌｅｔｅｄ）状態を有し、その寿命のあいだこれらのいずれかの状態をとる。

【０１７０】ドラフト状態

エントリは最初、データベースの境界の外側、かつＪａｖａのオブジェクト・ヒープの内部に存在するドラフト状態に構成される。エントリの作成者は、

```
SystemEntry sample = new SystemEntry("SampleName")
```

のようなコード断片を使用して名前の付いたエントリを構成する。この状態のエントリになされた変更に対してはイベントは生成されない。そのためエントリの作成者は自由に、属性を追加、変更、削除したり、所望の属性セットを達成したり、後に、データベースの残りの部分に対してエントリを公表したりすることができる。エントリの階層全体をオフラインで構築し、これを、サブツリーとして１回の操作で挿入することができる。公表エントリまたは公表エントリの階層をＪＳＤから分離すること、およびこれらを、オフラインであるとみなされるドラフト状態に戻すことができる。しかし、分離されたエントリへの参照が保持されるので、それらは真の意味でのオフラインではない。このようなエントリの変更を反映するイベントは生成されないが、ドラフト・エントリであっても世代番号は更新される。

【０１７１】公表状態

エントリは、データベース内の公表親エントリの下に挿入されると、公表状態に移行する。エントリがデータベースに入れられると、関係アプリケーションおよび／またはシステム・サービスは、エントリの名前および／または属性を検索基準として使用してエントリの場所を見つけることができる。エントリが挿入されると、ＪＳＤは挿入イベントを生成する。新しいエントリの調査に関係するリスナには、この方法で通知される。さらに必要に応じて属性イベントが、全ての公表エントリに対してＪＳＤによって生成される。

【０１７２】削除状態

エントリの削除は、オフラインのドラフト状態またはオンラインの公表状態のエントリをＪＳＤから除去することによって実施される。削除されたエントリに対する全てのメソッド呼出しは失敗となり、ＥｎｔｒｙＤｅｌｅｔｅｄＥｘｃｅｐｔｉｏｎが送出される。削除されたエントリは本質的にガベージ・コレクションを待つが、削除されたエントリへの参照を保持することができるので、それらへの参照が全て解放されるまで、それらの決定的な振舞いが必要となる。削除されたエントリは、他のどの状態になることもできない。

【０１７３】状態遷移

図２５に、エントリがその寿命の間に経過する可能性がある状態遷移を示す。番号をつけた遷移の定義は以下のとおりである。

１．エントリが構成される。

２．エントリが公表親エントリの下に挿入される。ＥｎｔｒｙＩｎｓｅｒｔＥｖｅｎｔが生成される。

３．公表子エントリが、（公表）親エントリから分離される。ＥｎｔｒｙＤｉｓｃｏｎｎｅｃｔＥｖｅｎｔが生成される。

４．公表エントリが削除される。ＥｎｔｒｙＲｅｍｏｖｅＥｖｅｎｔが生成される。

５．ドラフト・エントリが削除される。

６．ドラフト・エントリがドラフト親エントリの下に挿入される。

７．ドラフト子エントリが（ドラフト）親エントリから分離される。

【０１７４】エントリの操作

ＪＳＤエントリに対しては３つの基本操作、すなわち挿入、分離、および削除が定義される。

【０１７５】挿入

エントリはドラフト状態に構成される。ドラフト・エントリはドラフト親エントリの下、または公表親エントリの下に挿入することができる。前者の場合、この子エントリはドラフト状態のままであり、後者では、公表状態に変化する。公表状態では、子エントリはその親エントリのマネージャを継承する。この手順は、単一のエントリに適用されるのと同じように、（公表親エントリの下に挿入された）エントリのサブツリーにも適用される。サブツリー内のそれぞれのエントリに対して、ＥｎｔｒｙＩｎｓｅｒｔＥｖｅｎｔが生成され、それぞれのエントリは親のマネージャを継承する。挿入は、親エントリにｉｎｓｅｒｔ（）メソッドを呼び出し、それに子エン

トリへの参照を与えることによって実行される。親エントリは、子エントリの適性を判定し、その挿入を拒否することができる。例えば、親エントリがエイリアス（ＳｙｓｔｅｍＡｌｉａｓＥｎｔｒｙ）である場合、エイリアス・エントリは子エントリをサポートしないので挿入は失敗に終わる。同様にエントリが、あるタイプの子エントリしか受け入れない場合、またはその他の形態の家族計画に固執する場合がある。図２６参照のこと。

【０１７６】分離

エントリ（またはサブツリー）を、ドラフト親エントリまたは公表親エントリから分離することができる。親エントリのｄｉｓｃｏｎｎｅｃｔ（）メソッドが呼び出され、分離する子エントリへの参照が与えられる。その子エントリの子孫エントリも全て同様に分離される。公表エントリのみを分離するときには、ＥｎｔｒｙＤｉｓｃｏｎｎｅｃｔＥｖｅｎｔがそれぞれのエントリに対して生成される。親子関係が分離されるのは、分離された点においてのみである。子エントリの子孫エントリは全て、それらの家族関係（サブツリー）を維持する。サブツリー内のエントリは全てドラフト状態に変更され、それらのマネージャへの参照は維持される。図２７を参照のこと。ＰｅｒｓｉｓｔｅｎｔＳｙｓｔｅｍＥｎｔｒｙおよびＰｅｒｓｉｓｔｅｎｔＭａｎａｇｅｒによって提供される持続性実装は分離をサポートしない。マネージャへの参照が維持されるのは、公表親エントリから分離されたエントリへの参照が保持されている場合には、それらが依然としてアクセス可能である可能性があるからである。したがって、マネージャ参照は、分離されたエントリに実施されたメソッド呼出しを管理し、これらのエントリは、新しく構成されたドラフト・エントリほど原始的（ｐｒｉｓｔｉｎｅ）にはならない。ドラフト親エントリからドラフト・エントリのサブツリーを同じように分離することができる。この場合、手順は同一であるがイベントは生成されない。

【０１７７】削除

エントリ（またはサブツリー）の削除には、ＪＳＤのトランザクション性が与えられた２段階のプロセスが関与する。失敗の場合に全ての操作をロールバックすることができなければならず、したがってエントリ削除の第１の段階はエントリの分離になる。分離が成功した場合には、トランザクションのコミットに続いて実際の削除が実施される。サブツリー除去の最終段階を示す図２８参照。ガベージ・コレクションを促すために、削除されたエントリによって保持された全ての参照はヌルにセットされる。これには、全ての家族参照、マネージャ、クッキー、ロック、ならびにエイリアスの場合のエイリアシングされたエントリ参照などが含まれる。それらが、ＪＳＤの公表エントリであった場合には、分離されたエントリと同様に、削除されたエントリへの参照をそのまま維持し続けることができる。削除されたエントリについてメソッドを呼び出すと、ＥｎｔｒｙＤｅｌｅｔｅｄＥｘｃｅｐｔｉｏｎが送出される。これは、その参照の保持者に、それを解放し、削除されたエントリをガベージ・コレクションの対象とすることを指示する。

【０１７８】エントリ・イベント

システム・データベースは、データベースの変更を指示するイベントを生成する。全てのデータベース・イベントは、ＳｙｓｔｅｍＤａｔａｂａｓｅＥｖｅｎｔベース・クラスから派生する。したがって、そのイベント・オブジェクトのクラスは、受け取ったイベントのタイプを指示する。イベント・オブジェクトｇｅｔＥｎｔｒｙ（）メソッドは、影響を受けたデータベース・エントリへの参照を返す。Ｊａｖａ　Ｓｙｓｔｅｍ　Ｅｖｅｎｔ（ｊａｖａｘ．ｓｙｓｔｅｍ．ｅｖｅｎｔｓ）サブシステムは、イベントの生成者と消費者とを突き合せ、必要に応じてイベントを送る機構として使用される。ＪＳＥは、イベント・オブジェクトを用いてターゲット消費者を改善するのにＪＳＤが使用するイベント・フィルタ文字列を与える。ＪＳＤイベントは、そのイベントが関係するエントリまでのパス名を表す文字列を含む。ＪＳＥフィルタを、イベントの送達をさらにリファインするのに使用することができる。ＪＳＤイベントに関係する消費者は、ＪＳＥに直接に名前を登録する。この時点で、イベント登録のためのＪＳＤ　ＡＰＩはない。ＪＳＤは、全てのＪＳＤイベント・クラスの独占的生成者としてＪＳＥに名前を登録する。そのため、その他のソースがＪＳＤイベントを生成することはできない。イベント・フィルタリングを用いると、例えばイベント消費者だけが、／ｓｏｆｔｗａｒｅ名前空間内のエントリに関係するイベントに関与することができるようになる。このリスナは、フィルタ文字列「／ｓｏｆｔｗａｒｅ」を登録することができ、「／ｓｏｆｔｗａｒｅ」で始まるフィルタ文字列を含んでいるイベントだけが消費者に送達される。消費者への不必要なコンテキスト切替えを排除するため、このフィルタリングはイベント生成者のコンテキストで実行される。ＪＳＥイベント・フィルタリング機構は事実上の汎用性を有する。これは、スコーピング・イベントに備えるために特にＪＳＤによって使用され、消費者は、それらが関係する範囲のイベント・クラス・タイプをそれらに送る。スコーピングは汎用ではない。このフィルタ文字列は、プレフィックス（ｊａｖａ．ｌａｎｇ．Ｓｔｒｉｎｇ．ｓｔａｒｔｓＷｉｔｈ（））としての働きだけをする。一般的な正則表現等は現在のところサポートされていない。したがって、イベント範囲を設定することはできず、そのため、例えば「／ｔｅｍｐ」、「／ｃｏｎｆｉｇ」の中のイベントが関係するとみなされる。範囲がヌルまたは「／」であると、指定されたクラスの全てのイベントが送達される。

【０１７９】ＪＳＤの主要なイベント・カテゴリは２つある。

・データベース階層に関係するもの（エントリの挿入、分離、除去など）
・エントリの属性の変更の指示に使用されるもの（属性の追加、除去、変更など）
ＳｙｓｔｅｍＤａｔａｂａｓｅＥｖｅｎｔまたはそのサブクラスの１つをサブクラス化することによって、新しいイベント・カテゴリを将来、導入することができる。ＪＳＥは、イベント消費者が、Ｊａｖａクラス階層を利用することに備えている。すなわち、ＥｎｔｒｙＰｒｏｐｅｒｔｙＥｖｅｎｔを聴取（ｌｉｓｔｅｎ）することによって、リスナは、ＥｎｔｒｙＰｒｏｐｅｒｔｙＥｖｅｎｔタイプのイベントおよびその全てのサブクラスを受け取る。その結果、指定のスコーピングを受けた、属性変更に関係する全てのイベント（エントリの挿入、分離、および除去には関係しない）が受け取られる。

【０１８０】イベント・クラス階層
ＪＳＤのイベント・クラス階層を図２９に示す。ＳｙｓｔｅｍＤａｔａｂａｓｅＥｖｅｎｔの聴取の結果、全てのデータベース・イベントが受け取られる。イベント・クラス・タイプは、Ｊａｖａのｉｎｓｔａｎｃｅｏｆオペレータを介して区別することができる。ＥｎｔｒｙＥｖｅｎｔおよびその全てのサブクラスは、影響を受けたエントリへの参照を含む。ＥｎｔｒｙＰｒｏｐｅｒｔｙＥｖｅｎｔおよびそのサブクラスに、影響を受けた属性の名前を問い合わせることができる。以前の属性値（オブジェクト参照）は、ＥｎｔｒｙＰｒｏｐｅｒｔｙＲｅｍｏｖｅＥｖｅｎｔおよびＥｎｔｒｙＰｒｏｐｅｒｔｙＶａｌｕｅＣｈａｎｇｅＥｖｅｎｔから使用可能である。イベント・リスナとして登録するマネージャは、イベントを受け取ると適当なアクションをとることができる。例えばＰＣＭＣＩＡマネージャは、挿入および除去イベントを聴取し、ユーザがＰＣＭＣＩＡカードを挿入または取り出すときに適当に行動し、関連サービスをロードまたはアンロードし、カードに関係するインタフェース・エントリを追加または除去することができる。１つのレベルのリスナによるイベント応答によって、別のリスナにアクションを起こさせるイベントが生成されることがある。例えば、ＰＣＭＣＩＡモデム・カードを挿入することによって、ＰＣＭＣＩＡマネージャが、そのカードに必要なサービス（ドライバなど）をロードし、Ｉｎｔｅｒｆａｃｅ名前空間内にエントリを作成する。これによって、ダイヤラ・アプレットなどのより高いレベルのサービスをｉｎｔｅｒｆａｃｅ名前空間マネージャが見つけようとする別のイベントが生成される。

【０１８１】第４章　トランザクション
トランザクションは、ＪＳＤの１つまたは複数の関係エントリを原子的にロックしたり、書き込んだりする手段である。Ｔｒａｎｓａｃｔｉｏｎクラスは、エントリへの書込みアクセスのために排他的に実装される。すなわち、トランザクション・オブジェクトは、エントリへの全ての書込みアクセスに必要であるが、読取りアクセスには必要ない（ただし読取りアクセスが、トランザクション・エントリ・ロックに影響を与えることがある）。エントリの任意の集合を、トランザクションに関与させることができる。

【０１８２】サブツリー・トランザクション
ＪＳＤトランザクション・モデルはデータベースのサブツリー上で動作する。図３０にサブツリーを示す。サブツリーは、黒く塗りつぶしたエントリで示されている。このサブツリーは、ルート・エントリによって定義され、その子孫エントリの階層を通って全ての葉エントリまで延びる。トランザクションは、後に「機構」の項で説明するエントリ・ロックを獲得する。書込みアクセスでは、ロックは、単一のトランザクション・オブジェクトに排他的に関連づけられる。排他ロック操作が試みられるとき、それを許可するかどうかの決定は、所有トランザクションのａｌｌｏｗＷｒｉｔｅ（）メソッドに委ねられる。Ｔｒａｎｓａｃｔｉｏｎの現行の実装では、エントリ・ロックへのアクセスが、所有トランザクションを構成したスレッドだけに制限されている。これによって、エントリ・ロックはスレッドの細分性を与える。しかし所有スレッドは、ロックの再入が許されている。この方針は将来、トランザクションのメンバーへのマルチ・スレッド（おそらくはスレッド・グループ）同時アクセスができるように改められる可能性がある。しかし、ＪＳＤの最初のリリースにそれだけの複雑さは必要ない。

【０１８３】トランザクションの定義
ＪＳＤは、分散したフラットなトランザクション・モデルを提供する。トランザクションは、［１］に定義されているように、４つの属性、すなわちＡＣＩＤ属性を提供しなければならない。
１．原子性（Ａｔｏｍｉｃｉｔｙ）。トランザクションに関連した全ての状態変化が起こらなければならないか、または状態変化が一切起こってはならない。
２．整合性（Ｃｏｎｓｉｓｔｅｎｃｙ）。コミットされたトランザクションが状態を正確に変換する。
３．分離（Ｉｓｏｌａｔｉｏｎ）。トランザクションはコヒーレント状態を保証しなければならない。複数のトランザクションが状態を同時に修正することはできない。ただし、状態を直列に修正することはできる。
４．耐久性（Ｄｕｒａｂｉｌｉｔｙ）。トランザクションが正常終了した場合には、その後にどんな障害（システム・クラッシュなど）が起きても状態変化は維持される。
クライアントでは、メモリ・ベースのローカル・トランザクションは耐久性ではない。分散トランザクション・モデルは、持続性をサポートするのに必要である。Ｔｒａｎｓａｃｔｉｏｎクラスは、従属分散トランザクション識別子を定義できる。持続性実装は従属トランザクシ

ョンを利用し、ＡＣＩＤ属性が、エンド・ツー・エンドのフィーチャとなり、全体のＪＳＤトランザクション機構が持続性エントリに対するものとなるようにしなければならない。持続性エントリ用のネットワーク・コンピュータ基準実装の最初のリリースでは、ＡＣＩＤ属性の「Ｄ」（耐久性）が、ホスト・オペレーティング・システムのファイル・システムを介した基礎をなすＢｔｒｅｅへの書込みの正常終了までに制限される。

[1] Transaction Processing: Concepts and Techniques, Jim Gray, Andreas Reuter, Morgan Kaufmann Publishers, 1993. ISBN 1-55860-190-2.

【０１８４】トランザクション使用の概要

ＪＳＤトランザクションの使用には以下の３つの段階が関与する。

１．Ｔｒａｎｓａｃｔｉｏｎオブジェクトを構成する。サブツリーのルートは、コンストラクタへの引き数として指定される。構成中、サブツリーの中の全てのエントリは、トランザクションによって排他的にロックされる。持続性機構、すなわちローカル・ファイルまたはリモート・サーバが関与する場合には、コンストラクタが例外を送出せずにリターンした後に、必要な全てのロックがエンド・ツー・エンドで実施される。

２．ロックされたエントリに、エントリ属性の挿入、分離、削除、または修正などの操作を、Ｅｎｔｒｙ ｉｎｓｅｒｔ（）、ｄｉｓｃｏｎｎｅｃｔ（）、ｒｅｍｏｖｅ（）、ｒｅｍｏｖｅＰｒｏｐｅｒｔｙ（），およびａｄｄＰｒｏｐｅｒｔｙ（）メソッドを介して実行する。

３．ｃｏｍｍｉｔ（）メソッドを呼び出してトランザクションをコミットするか、またはａｂｏｒｔ（）メソッドを介してトランザクションをロールバックする。これらのメソッドは、トランザクションによって保持された全てのエントリ・ロックを解除する。ａｂｏｒｔ（）メソッドは、トランザクションをロールバックし、トランザクション・オブジェクトを無効にし、それ以後、これを使用できなくする。これは、エンド・ツー・エンド手順であり、そのため、持続性のために必要な従属トランザクションもコミットされるかまたは打ち切られる。ＪＳＤエントリへのアクセスは、エントリの探索およびそれらの属性値の読取りに最適化される。少なくともＪＳＤを初期化した後は、ＪＳＤアクセスの大部分がこのタイプになることが予想される。このようにエントリ・ロック機構（ＳｙｓｔｅｍＤａｔａｂａｓｅＬｏｃｋ）は複数のリーダを収容する。読取りは、トランザクション・ベースではないが、原子性が重要な場合には、いくつかのエントリ読取りロックを獲得することができる。しかし近い将来、デッドロック回復プロセスを援助するために、ＪＳＤで、読取りがトランザクションの対象となる可能性がある。ＪＳＤへの全ての書込みアクセスにはトランザクション・オブジェクトが必要である。トランザクションに関与するそれぞれのエントリに対して、排他ロックを獲得しなければならない。これによってオーバヘッドが生じる。トランザクションは楽観的であり、トランザクションの進行中にＪＳＤに実際に変更が実施される。トランザクションが打ち切られるまれなケースにのみ、潜在的に時間消費性のロールバックが実施される。

【０１８５】機構

エントリ、エントリ・ロック、およびトランザクションは非常に絡み合っている。エントリはその危険地域を定義し、エントリ・ロックが適正に使用されるようにチェックする。ｇｅｔＰｒｏｐｅｒｔｙＶａｌｕｅ（）、ｇｅｔＰｒｏｐｅｒｔｙＣｏｕｎｔ（）メソッドなどのメソッドを用いてエントリに読取りアクセスするとき、エントリは、属性リストにアクセスする前に読取りアクセスにロックされ、メソッドがリターンする前にロック解除される。読取りアクセスは現在、トランザクション・オブジェクトを含まないが、読取りロックによって、トランザクション・ベースの書込みロックは待機状態となる。排他アクセスが許されたときにのみ、書込みロックは成功する。エントリの状態を変更する（ｉｎｓｅｒｔ（）、ｄｉｓｃｏｎｎｅｃｔ（）、ｒｅｍｏｖｅ（））か、あるいは属性値を削除または変更する（ｒｅｍｏｖｅＰｒｏｐｅｒｔｙ（）、ａｄｄＰｒｏｐｅｒｔｙ（））全てのエントリ・メソッドは、トランザクション・オブジェクトを必要とする。これらのエントリ・メソッドは、トランザクションがエントリをロックしていることを確認してから、先に進む。ロックされていない場合には、ＥｎｔｒｙＬｏｃｋＳｔａｔｅＥｘｃｅｐｔｉｏｎが送出され、メソッドは失敗に終わる。エントリ・ロックはアクセスの獲得をいつまでも待つわけではない。ＪＳＤは、エントリにアクセスするパブリックＡＰＩ（すなわちエントリ読取りロック・メソッドｌｏｃｋＲｅａｄ（）およびｕｎｌｏｃｋＲｅａｄ（））およびＴｒａｎｓａｃｔｉｏｎクラスを提示するので、故意または偶然のＪＳＤデッドロックは回復可能でなければならない。デッドロックを確実に防ぐことはできず、そのため、検出と回復が唯一のオプションとなる。エントリ・ロックへの（読取りまたは書込み）アクセスの獲得を待っている間にスレッドがタイムアウトすると、デッドロックが検出される。これが起こると、ロックを所有するトランザクションのｓｙｓＡｂｏｒｔ（）メソッドが呼び出される。これによって、所有トランザクションが（ロールバックされて）打ち切られ、それが保持するロックが解除される。従属トランザクションも全て打ち切られる。トランザクション・オブジェクトは無効となり、将来にこれを使用しようとすると、ＴｒａｎｓａｃｔｉｏｎＩｎｖａｌｉｄＥｘｃｅｐｔｉｏｎが送出される。ロックがタイムアウトとなったスレッドに対しては、ＥｎｔｒｙＬｏｃｋＴｉｍｅｏｕｔＥｘｃｅｐｔｉｏｎが送出される。この時点でそれは、ロックの獲得を

再試行すべきか、またはこの操作を打ち切るべきかを判断することができる。デッドロックを引き起こしたトランザクションに対しては、割込み源トランザクションのオーナに混乱を収拾させるのではなく、ＪＳＤ自体が、そのロックのロールバックおよび解除を誘導することに留意されたい。これによって、ＪＳＤが有効な状態に戻ることが保証される。トランザクション・オブジェクトのユーザには、トランザクション・オブジェクトを使用した将来のメソッド呼出しによって送出されるＴｒａｎｓａｃｔｉｏｎＩｎｖａｌｉｄＥｘｃｅｐｔｉｏｎによって通知される。相互作用を示す図３１を参照。

【０１８６】例外

トランザクションおよびエントリ・ロックに関係するＪＳＤの例外階層を図３２に示す。ＴｒａｎｓａｃｔｉｏｎＩｎｖａｌｉｄＥｘｃｅｐｔｉｏｎは、無効なトランザクション・オブジェクト（打ち切られたトランザクション・オブジェクト）が、エントリに提示された（例えばａｄｄＰｒｏｐｅｒｔｙ（））とき、またはそのトランザクションにメソッド呼出しが実施された場合（例えばｃｏｍｍｉｔ（））に送出される。このイベントは、トランザクションのユーザに、それをもう使用できないこと、ＪＳＤの一切の変更がすでにそのためにロールバックされたことを知らせる。再試行するためには、新しいトランザクション・オブジェクトを構成する必要がある。ＥｎｔｒｙＥｘｃｅｐｔｉｏｎクラスは、全てのエントリ特定例外のスーパークラスである。これは、例外が関係するエントリへの参照を返すメソッドｇｅｔＥｎｔｒｙ（）を提供する。ＥｎｔｒｙＬｏｃｋＥｘｃｅｐｔｉｏｎは、エントリ・ロックに関係する全ての例外に対して同じ役割を果たす。エントリ・ロック例外は以下のように定義される。

【０１８７】ＥｎｔｒｙＬｏｃｋＳｔａｔｅＥｘｃｅｐｔｉｏｎ

トランザクションによって所有されていないロックを解除しようとするか、または、リーダ・カウントのアンダフローの発生によって生じる。このイベントを受け取る理由にはこの他、提供されたトランザクションがエントリ・ロックをすでに所有していることを期待するエントリ・メソッド（例えばａｄｄＰｒｏｐｅｒｔｙ（））が呼び出されたが、ロックを所有していない場合がある。エントリは構成時に、トランザクションの一部となっていなければならない。

【０１８８】ＥｎｔｒｙＬｏｃｋＴｉｍｅＯｕｔＥｘｃｅｐｔｉｏｎ

タイム・アウトは、トランザクション・コンストラクタを介して書込みロックを獲得するのを待っている間、またはＥｎｔｒｙ　ｌｏｃｋＲｅａｄ（）メソッドを介して読取りロックを獲得するのを待っている間に起こる。これは、デッドロックの検出を指示する。

【０１８９】トランザクション・オブジェクトの詳細

トランザクション・クラスはイベント待ち行列（ＪＳＥＯＳＥｖｅｎｔＱｕｅｕｅ）、およびトランザクションが保持するロックを記録するスタックを定義する。ロックは、トランザクション・オブジェクトを構成する間に指定されたエントリに対して獲得される。ロックされたエントリに修正が加えられている間、ＪＳＤイベントは、トランザクション・イベント待ち行列の中に維持され、トランザクションがコミットされるまでイベント消費者はこれを使用することができない。イベント待ち行列およびその他の補助的ＳｙｓｔｅｍＥｎｔｒｙフィールドを使用して、トランザクションをロールバックすることができる。この場合、イベント待ち行列は、トランザクション中に実行された操作の記録の役目を果たす。トランザクションがコミットされると、待ち行列中のイベントは、関係イベント消費者に配布される。この代わりにトランザクションが打ち切られた場合には、待ち行列を使用して、すでに実施された変更が取り消される。ロールバックが済むと、待ち行列中のイベントは掲示されなくなる。パブリック・コンストラクタは１つの引き数、すなわち、トランザクションのあいだロックされるデータベースのサブツリーのルートを示すエントリを受け取る。この時点で、イベント待ち行列およびロック・スタックが作成される。それはさらに、カレント・スレッドのＩＤを記録し、トランザクションの状態を有効にセットする。

```
public Transaction(Entry subTreeRoot) throws SystemDatabaseException;
```

トランザクションは、以下のメソッドを使用して所有スレッドによってコミットされるか、または打ち切られる。

```
final public void commit() throws SystemDatabaseException;
final public void abort();
```

トランザクションの状態は、

```
final public boolean isValid()
```

を介して問い合わせることができる。残りのメソッドは、プライベート・パッケージであり、分散管理を含めて、トランザクションの管理にＪＳＤの内部で使用される。さらに、ＳｙｓｔｅｍＥｎｔｒｙメソッドは更新の進行中にトランザクションにイベントを通知する。

【０１９０】トランザクションの完了

トランザクションは、以下に示す４つの方法のいずれかで完了となる。

１．トランザクションを使用するスレッドが、トランザクション（および従属分散トランザクションがローカルに）が失敗したことを指示する例外を受け取っていない。この場合、スレッドは、関連イベントをポストするｃｏｍｍｉｔ（）メソッドを呼び出し、トランザクションが保持する全てのロックを解除する。

２．ＪＳＤが、所有スレッドに失敗例外を送出しない場

合でも、スレッドが、（その他の高位の理由で）トランザクションの打切りを決定することがある。この場合にスレッドは、その変更（および従属トランザクションの変更）をロールバックするａｂｏｒｔ（）メソッドを呼び出し、イベント待ち行列を無効にし、全てのロックを解除する。トランザクション・オブジェクトは無効状態となり、それ以降それを使用することができない。

３．一般にデッドロック回復アクションの一部としてＪＳＤ自体が、ｐａｃｋａｇｅ　ｐｒｉｖａｔｅ　ｓｙｓＡｂｏｒｔ（）メソッドを使用してトランザクションを打ち切る。所有スレッドによる打切りと同様に、トランザクションはロールバックされ、イベント待ち行列は無効化され、全てのロックは解除される。トランザクションは無効状態となる。これは、所有スレッドとは非同期に実施されるので、所有スレッドが、このトランザクション・オブジェクトを将来に使用しようとすると、ＴｒａｎｓａｃｔｉｏｎＩｎｖａｌｉｄＥｘｃｅｐｔｉｏｎが送出されて、システム主導の打切りが所有スレッドに通知される。

４．トランザクションを所有するスレッドが終了している場合、トランザクションは、上記３に記載した方法でｓｙｓＡｂｏｒｔ（）を介して打ち切られる。

【０１９１】コミット

トランザクション中に変更が実施されるとそれらはローカルＪＳＤに加えられる。イベントが変更されると、それらは、トランザクション・イベント待ち行列にポストされる。トランザクションがコミットされると、待ち行列中のイベントが、（イベントが受け取られた順に）全ての関係消費者にポストされ、トランザクションが保持する全てのロックが解除される。キー・トランザクション・コンポーネントでは、エントリ（またはサブツリー）がＪＳＤから削除されるとき、エントリはＪＳＤから分離されるだけである。トランザクションがコミットされたときにのみ、分離されたエントリ（サブツリー）は実際に削除される。分離されたエントリの削除に失敗が生じる可能性はない。

【０１９２】打切り

トランザクションが打ち切られると、ＪＳＤへの全ての変更はロールバックされ、その後、全てのロックが解除される。ロールバックは、複雑で、潜在的に時間消費的なプロセスであり、持続性が関与するときには特にそうである。楽観的にトランザクションを見ると、ロールバックはまれなイベントであると期待される。トランザクション・イベント待ち行列は、ロールバック・プロセスを制御する機構である。イベントは、後入れ先出し方式で待ち行列から削除される。この時点で、イベント待ち行列はイベント・スタックとして使用されている。削除されるエントリ（サブツリー）は、この時点では分離されているだけであり（上記参照）、簡単に再接続することができる。挿入されたエントリは分離され（ドラフト状態に戻り）、分離されたエントリは再接続される。属性に関係するイベントは古い属性値を含み、そのためそれらを復元することができる。挿入された（新しい）属性は削除される。

【０１９３】分散モデル

過渡エントリへのトランザクションの実行はクライアント側でローカルに起こる。その他のホストは関与せず、ローカル持続性実装も含まれない。このようなトランザクションはＡＣＩ特性を満たす。過渡エントリの耐久性は保証されない。ＡＣＩＤ特性が持続性エントリにあてはまることを保証するためには、持続性実装による作業が必要である。ＩＩＯＰの他にＪＳＤプロトコルを使用するネットワーク・コンピュータ基準のデフォルト持続性実装によって、エンド・ツー・ニヤ・エンド実装が行われる。先に述べたように、この実装は、ホスト・オペレーティング・システム・インタフェースを使用してファイルを書き込む。これを「ニヤ・エンド」とするのは、ファイル・システムのバッファ・キャッシュが関与し、現在のところ、サーバ上のトランザクションが、ディスク媒体へのビットの書込みが成功したことを検査するまでには拡張されていないからである。将来的にはそのように拡張される可能性がある。将来の持続性実装が同様のことを保証しない場合、持続性エントリのトランザクションはこれらの実装でＡＣＩＤを満たさない。

【０１９４】トランザクションの成功

図３３に、２つの持続性エントリ、ｅｎｔｒｙ１およびｅｎｔｒｙ２に関するトランザクションを示す。ｅｎｔｒｙ２は、ｅｎｔｒｙ１の子孫エントリである。クライアントでトランザクションが構成される（ｃｔ）。持続性機構の結果、サーバにトランザクションが構成され（ｓｔ）、エントリがエンド・ツー・エンドでロックされることが保証される。持続性実装は、サーバで使用するコアレセンス（ｃｏａｌｅｓｃｅｎｃｅ）アルゴリズムを考慮に入れて、クライアント・エントリをサーバ・エントリにマップすることができなければならない。番号付きの矢印は以下のことを表す。

１．持続性エントリへの参照が、クライアントのトランザクション・コンストラクタに与えられ、持続性機構によって、サーバ上にもトランザクションが構成され、クライアントからサーバへエンド・ツー・エンドでサブツリーがロックされる。

２．クライアントが、ｅｎｔｒｙ１の属性を追加／変更する。これによって、クライアントのｅｎｔｒｙ１のコピーが更新され、次いでサーバのｅｎｔｒｙ１のコピーも更新される。

３．クライアントが、ｅｎｔｒｙ２の属性を追加／変更する。これによって、クライアントのｅｎｔｒｙ２のコピーが更新され、次いでサーバのｅｎｔｒｙ２のコピーも更新される。

４．クライアントが、ローカル・トランザクションのコ

ミットを試みる。クライアントはまず、トランザクションのコミットをサーバに要求する。次いでクライアントもコミットする。

2つのトランザクション・オブジェクトがコミットされ、エントリ・ロックが解除される。Ｔｒａｎｓａｃｔｉｏｎクラスは、分散従属トランザクションを結び付ける識別子を維持する。これらの識別子を使用して、一貫性のある結果を得るのに必要な初期接続手順（ｈａｎｄｓｈａｋｉｎｇ）を実装するのは、持続性実装の責任である。

【０１９５】トランザクション失敗モード

開始クライアント・トランザクションは、従属サーバ・トランザクションの識別子を保持する。サーバ・トランザクションは、クライアント・トランザクションの識別子を保持する。これらの識別子およびローカル・タイムアウト機構は、トランザクション全体をエンド・ツー・エンドに打ち切り、ロールバックするのに不可欠である。

【０１９６】クライアント打切り

何らかの理由でクライアントがトランザクションを打ち切る場合には、サーバ・トランザクションも打ち切られなければならない。クライアントは、従属トランザクションの識別子を使用してサーバにトランザクションの打切りを命じる。全ての変更はロールバックされ、全てのロックが解除される。クライアントでは、この打切りプロセスによって、全てのローカル変更がロールバックされ、全てのローカル・ロックが解除される。サーバの持続性機構も、トランザクションに代わって、持続性記憶に書き込まれた変更をロールバックしなければならない。

【０１９７】サーバ打切り

従属サーバ・トランザクションを打ち切る場合には、サーバは、クライアント・トランザクションの識別子を与えてクライアントにトランザクションの打切りを命じる。前記のクライアント打切りのシナリオと同様に、ロールバックは、クライアントとサーバの両方で実施される。

【０１９８】コンタクトの喪失

トランザクションは、タイムアウト機構を実施する。タイムアウトするとトランザクションは打ち切られる。この打切りの間に、従属トランザクションも打ち切られる。トランザクション・オブジェクトは無効になる。後に、リモート識別子を介してトランザクションにアクセスする場合、それは失敗に終わり、その識別子にアクセスしているトランザクションはその時点で打ち切られる。2つの相互依存トランザクションを有する1クライアント、1サーバの構成では、以下の回復アクションが可能である。

1．リモート・トランザクションが開始されると、クライアントはサーバとコンタクトすることができない。ある時点で、クライアント・トランザクションがタイムアウトし、打切りとなる。サーバが依然として作動している（例えばネットワークの問題などで）場合、そのトランザクションもある時点でタイムアウトし、打切りとなる。その結果、クライアントとサーバの両方がロールバックされる。

2．分散トランザクションの処理がクライアントによって開始され駆動される。サーバ・トランザクションがタイムアウトし、打ち切られた場合、クライアントが、リモート・トランザクション識別子を次に利用しようとしたときに、クライアントにこのことが通知され、クライアントは、そのローカル・トランザクションを打ち切る。

3．コンタクトが失われている時間が短く、どちらのトランザクションもタイムアウトしていない場合、完全なトランザクションを先に進めることができる。

【０１９９】

コード例

複数の原子性更新を実行して、単一のエントリにアクセスする。

```
Transaction t = new Transaction(entry1);
try [
        entry1.addProperty(t, "name1", value1);
        entry1.addProperty(t, "name2", value2);
        entry1.removeProperty(t, "name3");
        t.commit();
] (catch TransactionInvalidException ex) [
        //再試行？ロールバック済み
]
```

3つの異なるエントリにアクセスし、原子的な方法でそれらを修正する。この例で、ｅｎｔｒｙ２およびｅｎｔｒｙ３はｅｎｔｒｙ１の子孫エントリである。

```
Transaction t = new Transaction(entry1);
try [
        entry1.addProperty(t, "name1", value1);
```

```
                entry2.addProperty(t, "name2",
                entry3 .getPropertyValue("name3"));
                t.commit();
] (catch TransactionInvalidException ex) [
                //再試行？ロールバック済み
]
```

より現実的な例は、その構成情報を更新するアプリケーションである。

```
Tree t = SystemDatabase.getTree();
try [
                Entry me = t.findEntry("/software/application/com.appsRus/
    killerappl") ;
                Transaction t = new Transaction(me);
                me.addProperty(t, "background", bgValue);
                me.addProperty(t, "foreground", fgValue);
t.commit() ;
] catch (EntryNotFoundException ex) [
                //おっとまだ適正にインストールしていなかった
] catch (TransactionInvalidException ex) [
                //再試行？ロールバック済み
]
```

【０２００】第５章　データベース・ナビゲーション　ツリー

任意のデータベース・エントリをツリーのルートに指定すること、および任意のデータベース・エントリがＴｒｅｅインタフェースからの一連のメソッドに応答することができる。ルートの子孫エントリは全てツリーの一部である。あるコンテクストを提供するため、ツリー・インタフェースを使用して、全てのツリーに共通のメソッドを定義する。これらには、以下のメソッドが含まれる。

・ルート・エントリ
・カレント・エントリ
・ツリー・ポピュレータ
・ファインド・アンド・ニュー

ツリーは、ルート・エントリからその全ての子孫エントリまでの階層全体を含む。ツリーの範囲を制限する深さの概念はない。ツリーの名前は、指定されたツリー・ルートの名前と同じであり、単一の名前空間に属する。ツリーの中のエントリを、ルートまたは指定されたカレント・エントリからのパス名によって参照することができる。「／」で始まるパス名は、ツリー・ルートを基準にしてエントリを指名する。その他の形態のパス名は全て、カレント・エントリを基準にしてエントリを識別する。カレント・エントリを取得し、設定するメソッドが提供されている。ツリーのカレント・エントリは、ＵＮＩＸオペレーティング・システムの現行作業ディレクトリと類似のものである。

【０２０１】ツリー・ポピュレーション

ＴｒｅｅＰｏｐｕｌａｔｏｒインタフェースは、所与のツリーをポピュレートする一般的な機構を提供する。将来のＪａｖａの（例えばセルラ電話などへの）組込みバージョンでは、予め定義された文字列のスタティック・テーブルからデータベースのセクションをポピュレートすることが望まれる。電話のような単純なプラットフォームは、一続きの文字列で簡単に表すことができるいくつかの周知の固定デバイスを有する。より複雑なシステムでは、例えばＯｐｅｎＢｏｏｔファームウェア、ホスト・オペレーティング・システム情報（ＮＴレジストリ）、フラッシュ・メモリなどに基づいてツリー・ポピュレータを実装することができる。

【０２０２】ファインド・アンド・ニュー（Ｆｉｎｄ　Ａｎｄ　Ｎｅｗ）

Ｔｒｅｅインタフェースは、ツリーの中のパス名ルックアップを開始するのに使用するメソッド（ｆｉｎｄＥｎｔｒｙ（））、およびツリーに新しいエントリを追加するのに使用するメソッド（ｎｅｗＥｎｔｒｙ（））を定義する。これらのメソッドは、ルート・エントリにｌｏｃａｔｅ（）メソッドを呼び出すのとほぼ同じ働きをする。

【０２０３】データベース・パス名

パス名は、データベースの中のエントリを識別する文字列である。パス名は、名前空間内のスーパールートまたは指定されたカレント・エントリを基準として解釈される。ＵＲＬをパス名として使用することについては検討中である。

【０２０４】例外

データベース内のエラーはＪａｖａ例外を介して伝達される。ＳｙｓｔｅｍＤａｔａｂａｓｅＥｘｃｅｐｔｉｏｎクラスは、全てのＪＳＤ関連例外のスーパークラスである。データベース・メソッドの大部分は、Ｓｙｓｔｅ

mDatabaseExceptionの送出によって宣言される。これは一般に、キャッチ・オール（catch all）表記である。特定のメソッドが送出することができる特定のデータベース例外についてはJavadocドキュメンテーションを参照されたい。JSD例外階層を示す図34参照。ほとんどの種類の例外は、特定のエントリへのメソッド呼出しに関するエラーに関係する。それぞれの例外クラスが送出される条件および使用可能なメソッドの詳細については「例外クラス」の項を参照されたい。

【0205】データベース・ナビゲーション

JSDは、エントリ参照およびパス名ルックアップを介したナビゲーションを提供する。エントリ参照は、EntryインタフェースのgetParent()およびgetChildEntries()メソッドによって実現される。パス名ルックアップは、Treeインタフェースによって定義されているfindEntry()メソッドによって提供される機能である。Queryクラスは、エントリ名および属性名に基づいてJSDのより複雑な探索を実行するために提供される。ただし最終的には、問合せの結果としてエントリ参照が返される。Entry getParent()メソッドは、そのエントリの親エントリへの参照を返す。エントリは、親エントリを1つだけ持つ。ドラフト状態のエントリは、親エントリを持つ場合と持たない場合がある。getChildEntries()は、列挙（enumeration）を返す。それぞれの子エントリへの参照をこの方法で獲得することができる。これらのメソッドは、データベース・パッケージ自体の内部で一般に使用されるが、JSDエントリを検出し、作成する他のより便利な機構が使用可能である。

【0206】Treeインタフェースは、findEntry()およびnewEntry()メソッドを定義する。これらのメソッドはともに、問題のエントリのパス名を表す文字列を受け入れる。パス名は、「/usr/bin/rm」などのようなUNIXスタイルのファイル名と同様である。例外EntryNotFoundException、EntryNameExistsException、およびEntryNameInvalidExceptionは失敗モードを指示するために送出される。Entryインタフェースは、パス名ルックアップのフックを提供する。locate()およびisBasename()メソッドがフックにあた＊る。locate()メソッドによって、データベース・エントリが、その子エントリの中に、そのパス名の対象エントリ（最終コンポーネントまたはベース名）、または対象エントリに到達するのに通過するコンポーネントとなっているエントリを含むかどうか指示する。LocateResultクラスのオブジェクトへの参照がlocate()に与えられる。このオブジェクトは、この呼出しの結果を指示するように修正される。有効なエントリ名およびパス名コンポーネント・セパレータを表す文字を明示的に指示しないことによって、JSDは柔軟になる。SystemEntryベース・クラスのパス名コンポーネント・セパレータは、フォワード・スラッシュ（「/」）である。このように、SystemEntryオブジェクトだけから成るパス名は非常にUNIXに似ている。しかし、Entryインタフェースのその他の実装は、異なるコンポーネント・セパレータ文字を選択したり、または名前をある型に限定したりすることができる。

【0207】図35に、仮定のx86ベースのクライアント上のJSDサブセットを示す。この例では、データベースのこの部分にあるエントリは全て、SystemEntryオブジェクトである。ブート時、スーパールート・エントリを作成した後に、SystemTreeオブジェクトが、データベースの最上位ツリーを示すようインスタンス化される。このツリーのルート・エントリはスーパールート・エントリである。図示の例では、「/device/i86pc/isa」で示されるエントリがツリーのカレント・エントリである。このエントリは、このマシンのISAバスを表す。ISAバス上のフロッピー・ディスク・コントローラへのエントリ参照を取得するためには、システム・データベース・ツリー・オブジェクトのfindEntry()メソッドを、パス名「/device/i86pc/isa/fdc」で呼び出すか、または、ISAバス・エントリがカレント・エントリとなっているのでパス名「fdc」で呼び出す。最初の文字がセパレータ文字でないパス名の基準は、SystemTreeオブジェクトに定義されたカレント・エントリとなる。パス名ルックアップが始まるエントリは、探索開始点（starting search point）として知られる。パス名ルックアップは以下のように進行する。

【0208】

```
public Entry findEntry(String path) throws
                    SystemDatabaseException [
                      entry = getStartingSearchPoint(path);
                    //locateメソッドがLocateResultクラスを介してパスを修正する
                      class while (path is not empty) [
                      //EntryNotFoundExceptionを送出することができる
                        entry = entry.locate(remaining#path);
```

return entry;

【0209】図35の例を使用して、パス名「／ｄｅｖｉｃｅ／ｉ８６ｐｃ／ｉｓａ／ｆｄｃ」を与えると、探索開始点はスーパールート・エントリになる。スーパールートでｌｏｃａｔｅ（）が呼び出され、残りのパスが「ｄｅｖｉｃｅ／ｉ８６ｐｃ／ｉｓａ／ｆｄｃ」になる。「ｄｅｖｉｃｅ」という名前の子エントリがあるため、ｌｏｃａｔｅ（）は成功し、一致した子エントリへのエントリ参照が返される。次いでパス名が「ｉ８６ｐｃ／ｉｓａ／ｆｄｃ」になる。「ｄｅｖｉｃｅ」エントリのｌｏｃａｔｅ（）メソッドが呼び出され、以下同様の操作が実施される。最後に、「ｉｓａ」という名前のエントリのｌｏｃａｔｅ（）メソッドが呼び出され、「ｆｄｃ」という名前の子エントリを見つける。この時点で残りのパスはなく、最後のパス名コンポーネントのマッチングが成功したことが分かる。パス名コンポーネントが、そのエントリの子エントリの１つと一致するかどうかを指示することに加え、ｌｏｃａｔｅ（）メソッドは、一致したコンポーネントを削除してパス名を更新する。そのため、前記のｗｈｉｌｅループは最後には終了する。この機構の利点は、Ｅｎｔｒｙインタフェースの実装によって、パス・コンポーネントを含むものが定義されることである。この方針は、子エントリに代わってその親エントリによって実施される。

【0210】例えば、装着されたＮＦＳファイル・システムを表す仮定のクラスＮＦＳＥｎｔｒｙを与えると、ナビゲーションは、図36のこのエントリ（グレー）に対してどのような異なる振舞いを見せるだろうか。この例では、システム・データベース・ツリーのカレント・エントリは「ｖｏｌｕｍｅ」エントリであり、そのため、ＮＦＳマウント・ポイント・エントリには、「／ｉｎｔｅｒｆａｃｅ／ｖｏｌｕｍｅ／ｍｎｔｐｔ」または単に「ｍｎｔｐｔ」で到達することができる。エントリ「／ｉｎｔｅｒｆａｃｅ／ｖｏｌｕｍｅ／ｍｎｔｐｔ／ｄｉｓｔ／ｂｉｎ／ｐｒｏｇ」を見つける手順は、最初の例と同じように始まる。しかし、ｌｏｃａｔｅ（）メソッドが「ｍｎｔｐｔ」エントリで呼び出されたときには、このメソッドに、パス「ｄｉｓｔ／ｂｉｎ／ｐｒｏｇ」が与えられる。メソッドは、この残りのパスを使用して、ＮＦＳサーバから情報を獲得することができる。次いでＮＦＳＥｎｔｒｙオブジェクトは、指定されたファイルを参照するデータベース・エントリを作成することができる。あるいは、ＮＦＳＥｎｔｒｙ自体がある状態を維持し、（おそらくは残りのパスに基づいたハッシュテーブル・エントリを介して）ファイル自体を表すことができる。

【0211】所与のＥｎｔｒｙ実装を高度にカスタマイズできることは容易に分かる。専門化されたエントリのｌｏｃａｔｅ（）メソッドに、パス「ｍａｃｈｉｎｅ１．ｃｏｍ｜ｍａｃｈｉｎｅ２．ｃｏｍ：ＳＴＡＴＵＳ」を与えると、ｌｏｃａｔｅ（）メソッドは、この文字列を分割する方法を知り、例えば、これらの２つのマシンにステータスを問い合わせ、次いで、そのステータスを表すデータベース・エントリをインスタンス化する。パス名に基づいて新しいエントリを作成するときには、その文字列のベース名（最後のコンポーネント）が、新しいエントリの名前に使用される。少なくともＳｙｔｓｅｍＥｎｔｒｙではこれがあてはまる。ルックアップ・プロセスの間、個々のエントリのパス名構文は未知であるので、ＳｙｓｔｅｍＴｒｅｅ　ｎｅｗＥｎｔｒｙ（）メソッドは、親ＳｙｓｔｅｍＥｎｔｒｙの地点までのルックアップを実行することができる。残りのパスと一致する子エントリがない場合、新しいＳｙｓｔｅｍＥｎｔｒｙオブジェクトがインスタンス化され、先の残りのパスの名前がその名前となる。

【0212】しかし、残りのパスが例えば「ｘｘｘ／ｙｙｙ」であると、これは、ＳｙｓｔｅｍＥｎｔｒｙに対して無効となる。それはこれが、コンポーネント・セパレータを含むからである。ＳｙｓｔｅｍＴｒｅｅは、セパレータが何であるか、または、エントリ名にその他の制限があるかどうかを知らないので、親エントリのｉｓＢａｓｅｎａｍｅ（）メソッドを呼び出す。このメソッドは、残りのパスが子エントリに適当であるかどうかを指示する。適当である場合には、親エントリのマネージャを使用して、子エントリとして挿入される新しいインスタンスを作成することができる。以上に記載したパス名ルックアップ法は、外見は任意のＪＳＤルールを単に全てのエントリに実施する方法よりもいくぶん複雑で、実装にコストがかかるが、非常に大きなパワーと柔軟性を提供する。

【0213】データベース検索

この項では、データベース検索サービスの概要を説明する。ＪＳＤは、指定された基準と一致するエントリを求めてデータベースを検索するＱｕｅｒｙとＰｒｏｐｅｒｔｙＱｕｅｒｙと呼ばれるクラスを定義する。マッチング基準に、エントリ名または任意の属性名を含めることができる

Ｑｕｅｒｙオブジェクトは、検索開始点（エントリ）および検索基準を与えることによって構成される。問合せは、検索基準と一致するエントリを探すＪａｖａの列挙インタフェースをサポートする。検索をリセットしたり、バックアップする追加のメソッドも提供される。ＪＳＤの問合せには以下のものが含まれる。

・名前（文字列）
・検索範囲
・検索状態
・最後の一致エントリ
・現在の一致エントリ

【0214】検索基準

Ｑｕｅｒｙクラス・コンストラクタは、検索範囲内に定義された全てのエントリに対して突き合わせを行う名前文字列の指定を考慮する。ＰｒｏｐｅｒｔｙＱｕｅｒｙコンストラクタは、属性名を検索する同じ機能を提供する。

検索範囲

検索の範囲を、自己（ｓｅｌｆ）、親（ｐａｒｅｎｔ）、兄弟（ｓｉｂｌｉｎｇｓ）、子（ｃｈｉｌｄｒｅｎ）、または子孫（ｄｅｓｃｅｎｄａｎｔｓ）とすることができる。

自己

検索開始点のエントリのみを対象に、エントリ名または属性名を検索する。

親

検索開始点エントリの親エントリのみを対象に、エントリ名および属性名を検索する。

兄弟

検索開始点エントリ自体を含む全ての兄弟または検索開始点エントリを対象に、エントリ名および属性名を検索する。これらより下のエントリは検索しない。

子

検索開始点エントリの直接の子エントリのみを対象に、エントリ名および属性名を検索する。

子孫

検索開始点エントリの全ての子孫エントリ（サブツリー階層全体）を対象に、エントリ名および属性名を検索する。

検索結果

それぞれの一致エントリへの参照を獲得するメソッドが、ＱｕｅｒｙおよびＰｒｏｐｅｒｔｙＱｕｅｒｙクラスによって提供される。標準Ｊａｖａ　Ｅｎｕｍｅｒａｔｉｏｎインタフェースも実装される。

【０２１５】第６章　クライアント／サーバの要点

この章では、ＪＳＤクライアント／サーバ・アーキテクチャの概要をその高位レベルに限って説明する。現在出版されているもの、または近く出版されるものの中に、クライアント／サーバの同期をとるために使用される機構およびプロトコルをより詳細に述べたものがある。またこの章は、ブート・プログラム（ＰＲＯＭなど）が、ブート・サーバとコンタクトを確立する方法、コア・オペレーティング・システムをダウンロードする方法、または必要なプロトコルを呼び出す方法を説明しようとするものではない。

【０２１６】導入部

ＪＳＤは、痩せた（ｔｈｉｎ）クライアントの実装を容易にするクライアント／サーバ・モデルをサポートする。このようなクライアントは、ネットワーク上の１つまたは複数のサーバから、構成情報、サービス、アプリケーション、およびユーザ・データ・アクセスを獲得する。あるサーバは、クライアントをブートするのに必要な実行可能でロード可能なサービスをクライアントに提供し、別のサーバは、ユーザ・ログインを認証し、ユーザ特定構成情報をクライアントに提供する。また別のサーバは、アプリケーションおよびユーザ・データの送信元となる。将来、クライアント自体が、サーバ側ＪＳＤを実装し、モバイル（ポータブル）コンピューティングを提供する可能性がある。この場合、クライアントには、ある形態のローカル持続性記憶域（ディスク、フラッシュ・メモリ、バックアップ・バッテリ付きのＲＡＭなど）が必要となる。ネットワークに再接続されたときに、クライアントは、１つまたは複数のサーバと同期をとることができる。

【０２１７】サーバ・スキーマ

サーバには、システム・アーキテクチャの章で説明したものと同じ最上位レベル名前空間が存在する。クライアントでは、ｓｏｆｔｗａｒｅ名前空間が持続性であり、この名前空間には、そのハードウェア・プラットホームおよび現在ログインしているユーザに特定の情報が含まれる。サーバでは、ｃｏｎｆｉｇ名前空間が持続性である。クライアントは一度に、単一のマシン・アーキテクチャおよび一人のユーザをサポートすればよいのに対して、サーバは、複数のクライアントおよびユーザのための情報を維持しなければならない。その結果、ｃｏｎｆｉｇ名前空間は、大きなマシンおよびユーザ・カテゴリを含む。ｃｏｎｆｉｇ名前空間のサブスキーマを表す図３７参照。

【０２１８】マシン

サーバは、マシン固有識別子を介して特定のクライアントを識別する。これは一般に、ＭＡＣアドレス（イーサネット、トークンリング）およびハードウェア・タイプのようないくぶん低レベルのものである。識別子文字列が与えられると、サーバは、／ｃｏｎｆｉｇ／ｍａｃｈｉｎｅ／ｉｄｅｎｔｉｆｉｅｒｓ／ｍａｃｈ＿ｕｎｉｑｕｅ＿ｉｄの中で特定のクライアントの情報の場所を見つけることができる。所与のクライアント・ファミリー（例えばＳｕｎのＪａｖａＳｔａｔｉｏｎ）に共通の情報を、プラットフォーム・サブエントリ、例えば／ｃｏｎｆｉｇ／ｍａｃｈｉｎｅ／ｐｌａｔｆｏｒｍ／ＪＤＭ１の中で見つけることができる。マシン識別子エントリは一般に、プラットフォーム・エントリを参照して、共通情報を共用し、管理を容易にする。特定のクライアントが、プラットフォームに追加情報を提供するか、またはオーバライドするとシステム管理者がみなした場合、この情報は、そのマシン特定エントリに維持される。最後に、マシン固有エントリは、１つまたは複数のプロファイルを参照することができる。プロファイルは、属性を共有するマシンの追加のグルーピングを考慮する。例えば、小売状況では、スポーツウェア部門のクライアントは全て、宝石部門のクライアントが参照するプロファイルとは異なるプロファイルを参照することができる。

マシン・アーキテクチャ（プラットフォーム）はこのプロファイルを指図しないが、システム管理者が決定したその他の関係がこれを行う。マシン固有エントリ、そのプラットフォーム、およびプロファイルは協力して、クライアントがブートされた後、ユーザがログインする前にクライアントにダウンロードするサービス、アプリケーションなどを決定する。マシン階層を表す図３８参照

【０２１９】ユーザ

ユーザ・カテゴリは、マシンに提供される機能と同様の機能をユーザに提供する。ユーザは、「ｂｉｌｌ」などのログイン識別子によって識別される。ユーザ・エントリは、グループ・エントリを参照することができる。ただし参照しなくともよい。グループ・エントリは、１人または複数人のユーザに共通の情報を提供する。グループは、システム管理者によってセットアップされ、維持される。ユーザ・エントリは、システム管理者の干渉なしにユーザが属性をカスタマイズすることができる唯一の場所である。ユーザは、ログイン後に追加のサービスをロードすること、またはある属性値で、グループ・エントリに定義された属性値をオーバライドすることを指示することができる。許される場合には、ユーザ属性で、マシン・エントリ階層から得た属性をオーバライドすることもできる。ユーザ階層を示す図３９を参照。

【０２２０】サーバ・データ・コアレセンス

クライアントに最終的にダウンロードされる情報は、１つまたは複数のサーバのｃｏｎｆｉｇ名前空間の中のさまざまなソースに由来する。エントリおよびそれらの属性、ならびに結果としてロードされたサービスおよびアプリケーションはクライアントＪＳＤに現れる。データの使用順序を課すのはＪＳＤサーバの責任である。一般に、より特定な情報が、より一般的な情報にオーバライドする。サイト・セキュリティ方針を侵害しない限り、ユーザ情報はマシン情報に優先する。一般に、マシン特定情報が、関連プラットフォームまたはプロファイル情報にオーバライドする。プロファイルが他のプロファイルにオーバライドする方法は、システム管理者によって決められる。ユーザ特定情報は一般に、ユーザ・エントリが参照する１つまたは複数のグループに由来する情報にオーバライドする。この場合も、セキュリティー方針によって、これのいくつかまたは全てが起きることを防ぐことができる。同様に、ユーザ起源の情報が、マシン階層からの情報にオーバライドするようにしてもよいし、しなくてもよい。それにもかかわらず、マシン・データとユーザ・データのコアレセンス（ｃｏａｌｅｓｃｅｎｃｅ）がサーバで起こる。その結果は、クライアントのｓｏｆｔｗａｒｅ名前空間にダウンロードされる。クライアント・スキーマは、クライアント上で、全てのサービスおよびアプリケーションがそれらの構成情報を見つける機構である。

【０２２１】クライアント／サーバ通信

図４０に、ＪＳＤに持続性を実装するのに使用する高位の機構を示す。持続性エントリは、持続性マネージャ・クラスの助けを借りて、持続性記憶域に出し入れされる。一般的なクライアントの場合、持続性マネージャは、その持続性操作をＩＩＯＰを介してサーバに伝達する。サーバは一般に、大容量記憶デバイスとの相互作用で、持続性を実現する。しかしサーバの持続性マネージャは、ＪＳＤ／ＩＩＯＰプロトコルを使用して他のサーバと簡単に通信する。あるいは代わりに、ＮＩＳ、ＬＤＡＰ、ＡＣＡＰまたはその他のディレクトリ／命名サービスなどのその他の機構を使用することもできる。複数の持続性マネージャ・クラスを実装することによって選択の結合をサポートすることができる。持続性マネージャは、定義された持続性マネージャ・インタフェースを実装するだけでよい。基礎となる潜在的な実装の数は限りないが、その場合もクライアント／サーバの相互運用性を考慮しなければならない。例えば、クライアントの持続性マネージャが、持続性のためにＡＣＡＰプロトコルを使用するが、ＡＣＡＰサーバが使用できない場合、たとえいくつかのＪＳＤ／ＩＩＯＰプロトコル・サーバが使用できたとしてもそのクライアントにとって何の役にも立たない。ＪＳＤクライアントに、ＪＳＤ／ＩＩＯＰプロトコルを実装させることによって、この状況を改善することができるが、サーバは、使用可能な持続性記憶手段を実装することができる。持続性の情報をローカル・ディスクに持ち続けるか、ＡＣＡＰクライアントとして機能するか、または、ローカルＷｉｎｄｏｗｓ ＮＴレジストリにアクセスするかをサーバが選択する場合、それは、クライアントにとって関係の無いことである。

【０２２２】ＪＳＤプロトコル

持続性マネージャ・クラスは、高水準ＪＳＤプロトコルをサポートするために実装するメソッドを決める。これらのメソッドは、合体された（ｃｏａｌｅｓｃｅｄ）サーバ・サブツリーのダウンロード、クライアントが開始する変更のプッシュバック、クライアント・エントリの無効化などの操作を定義する。ある意味で、複数のクライアントが共通データを共用すると、キャッシュ・コヒーレンシ・プロトコルが実装される。その最初の実装では、クライアントとサーバの間でオブジェクトを前後に移動させるプロトコルとしてＩＩＯＰが使用される。ＬＤＡＰ、ＡＣＡＰなどのその他の機構を実装するために、この実装のこの部分を将来的に変更することができる。モバイル・コンピューティングについては既に述べた。クライアントがサーバにアクセスしているか否かに応じて異なる振舞いをする持続性マネージャ・クラスを実装することができる。まず、（おそらくはユーザがアプレットを走らせることによって）サーバから分離されることを見越して、クライアントは、オフラインの間にアクセスする必要がある追加のサービスおよびアプリケ

ーションをダウンロードすることができる。分離後には、クライアントによって開始されたｓｏｆｔｗａｒｅ名前空間への変更をローカル・キャッシュに入れることができる。サーバの存在を検出するとクライアントはその変更を、サーバで実施された変更と同期させる。持続性マネージャ・ベースクラスにこの機能を組み込むのは、サーバ・システム・クラッシュまたはネットワークの中断などの間に、サーバとのコンタクトが失われた場合のはね返りを見越したものである。

【０２２３】第７章　クラス／インタフェースの要点

この章では、ＪＳＤクライアントＡＰＩの高位の要点について述べる。キーとなるインタフェースおよびクラスのメソッド・シグナチャを示し、それらの目的を簡単に説明する。詳細についてはＪａｖａｄｏｃベースのドキュメンテーションを参照されたい。メソッドの大部分は、ＳｙｓｔｅｍＤａｔａｂａｓｅＥｘｃｅｐｔｉｏｎを送出する。この例外クラスは、ＪＳＤの最上位クラスである。ＪＳＤのその他の全ての例外は、Ｓｙｓｔｅｍ＊ＤａｔａｂａｓｅＥｘｃｅｐｔｉｏｎのサブクラスである。ＪＳＤの例外クラスの階層についてはこの章の「例外クラス」の項を参照されたい。さらに、メソッド引き数がイリーガルであるとみなされた場合（例えばヌル・エントリ参照）には、ｊａｖａ．ｌａｎｇ．ＩｌｌｅｇａｌＡｒｇｕｍｅｎｔＥｘｃｅｐｔｉｏｎを、ＪＳＤのセキュリティ・マネージャが、メソッド引き数がイリーガルであると判定した場合には、ｊａｖａ．ｌａｎｇ．ＳｅｃｕｒｉｔｙＥｘｃｅｐｔｉｏｎを送出することができる。ＪＳＤのクラスおよびインタフェースは、以下の４つのヘッデングの下にグループ分けされる。

・インタフェース
・クラス
・イベント
・例外

【０２２４】インタフェース

Ｅｎｔｒｙインタフェース

表１－１　Ｅｎｔｒｙインタフェース

メソッド	機能
public String getName();	エントリに関連した名前を返す。
public boolean isDrafted();	エントリがドラフト・エントリであるかどうかを指示する。
public boolean isPublished();	エントリが公表エントリであるかどうかを指示する。
public boolean isDeleted();	エントリが削除エントリであるかどうかを指示する。
public boolean isPersistent() throws SystemDatabaseException;	エントリが持続性である（サーバまたはローカル記憶デバイスにバックアップされている）場合に真を返す。
public long getGenerationNumber() throws SystemDatabaseException;	世代番号を返す。
public Entry getParent() throws SystemDatabaseException;	エントリの親エントリを返す。
public Enumeration getChildEntries() throws SystemDatabaseException.	エントリの子エントリへの参照を得るのに使用した列挙を返す。
public int getNumberOfChildren() throws SystemDatabaseException.	エントリの子エントリの数を返す。
public Manager getManager() throws SystemDatabaseException;	マネージャ参照を返す。
public Manager setManager(Manager mgr) throws SystemDatabaseException;	匿名のトランザクション・オブジェクトを使用してマネージャ参照を設定する。以前のマネージャ参照を返す。
public Manager setManager (Transaction t, Manager mgr) throws SystemDatabaseException.	マネージャ参照を設定する。

public void locate(LocateResult lr) throws SystemDatabaseException;	ＬｏｃａｔｅＲｅｓｕｌｔを見つけ更新する。
public Service assignService(Service newService) throws SystemDatabaseException;	エントリにサービスを割り当てる。匿名のトランザクション・オブジェクトを使用する。
public Service assignService (Transaction t, Service newService) throws SystemDatabaseException;	エントリにサービスを割り当てる。
public Service getAssignedService() throws SystemDatabaseException;	エントリに割り当てられたサービスを取得する。
public boolean isBasename(String name) throws SystemDatabaseException;	名前が、有効なベース名であるかどうかを指示する。
public void insert(Entry childEntry) throws SystemDatabaseException;	参照された子エントリを挿入する。匿名のトランザクション・オブジェクトを使用する。
public void insert(Transaction t, Entry childEntry) throws SystemDatabaseException;	参照された子エントリを挿入する。
public void disconnect(Entry childEntry) throws SystemDatabaseException;	参照された子エントリを分離する。匿名のトランザクション・オブジェクトを使用する。
public void disconnect(Transaction t, Entry childEntry) throws SystemDatabaseException;	参照された子エントリを分離する。
public void remove(Entry childEntry) throws SystemDatabaseException;	参照された子エントリを削除する。匿名のトランザクション・オブジェクトを使用する。
public void remove(Transaction t, Entry childEntry) throws SystemDatabaseException;	参照された子エントリを削除する。
public int getPropertyCount() throws SystemDatabaseException;	このエントリに関連した追加属性の名前および値のカウントを返す。
public Enumeration getPropertyNames() throws SystemDatabaseException;	これらの属性の列挙子を返す。
public boolean hasPropertyWithName (String name) throws SystemDatabaseException;	エントリが属性名「ｎａｍｅ」を有する場合に真を返す。
public Object getPropertyValue (String name) throws SystemDatabaseException;	属性名「ｎａｍｅ」を有する属性の値を取得する。
public Object addProperty (String name, Object value) throws SystemDatabaseException;	値「ｖａｌｕｅ」を有する「ｎａｍｅ」という名前の属性を追加する。匿名のトランザクション・オブジェクトを使用する。この属性の以前の値を返す

public Object addProperty (Transaction t, String name, Object value) throws SystemDatabaseException;	。 値「ｖａｌｕｅ」を有する「ｎａｍｅ」という名前の属性を追加する。この属性の以前の値を返す。
public Object addProperty(String name) throws SystemDatabaseException;	匿名のトランザクション・オブジェクトを使用してブール属性を追加する。この属性の以前の値を返す。
public Object addProperty (Transaction t, String name) throws SystemDatabaseException;	ブール属性を追加する。この属性の以前の値を返す。
public Object removeProperty(String name) throws SystemDatabaseException;	「ｎａｍｅ」という名前の属性を削除する。匿名のトランザクション・オブジェクトを使用する。その属性の値を返す。
public Object removeProperty (Transaction t, String name) throws SystemDatabaseException;	「ｎａｍｅ」という名前の属性を削除する。その属性の値を返す。

20

public void lockRead() throws SystemDatabaseException;	読み取るエントリをロックする。
public void unlockRead() throws SystemDatabaseException;	読み取るエントリのロックを解除する。
public void lockWrite(Transaction t) throws SystemDatabaseException;	書き込むエントリをロックする。
public void unlockWrite(Transaction t) throws SystemDatabaseException;	書き込むエントリのロックを解除する。
public void reenterWrite(Transaction t) throws SystemDatabaseException;	書き込むエントリがロックされているかどうかを再入する（検証する）。
public void buildPath(Tree tree, StringBuffer buf) throws SystemDatabaseException;	指定されたバッファの中にエントリの絶対パス名を構築する（ｗｒｔ　”ｔｒｅｅ’）。
void printEntry(PrintStream out);	デフォルトの記述法でエントリを印刷する。
void printEntry(PrintStream out, String prefix, boolean properties, boolean verbose, boolean debug, boolean children, boolean recurse);	指定された記述法でエントリを印刷する。

【０２２５】Ｔｒｅｅインタフェース

40

表１－２．Ｔｒｅｅインタフェース

メソッド	機能
public Entry getRootEntry();	ツリーのルート・エントリを返す。
public Entry getCurrentEntry();	指定されたカレント・エントリを返す。
public boolean setCurrentEntry (Entry nextCurrentEntry) throws SystemDatabaseException;	エントリを、カレント・エントリとして指定する。

50

73	74
public Entry findEntry (String pathName) throws SystemDatabaseException;	パス名が与えられたエントリを検索する。
public Entry newEntry (String pathName) throws SystemDatabaseException;	残りのパスによって識別された親エントリの下に、ベース・パス名を有する新しいエントリを作成する。
public Entry newEntry(Entry parent, String name) throws SystemDatabaseException;	新しいエントリを作成し、指定された親エントリの子としてそれを挿入する。
public boolean newEntry (String parent, Entry child) throws SystemDatabaseException;	指定された親エントリの子としてエントリを挿入する。
public Entry newAlias(String aliasPath, String aliasedEntryPath) throws SystemDatabaseException;	指定されたパス名のエントリを参照する、指定されたパス名の新しいエイリアスを作成する。そのエイリアスのエントリを返す。
public Entry newAlias(String aliasPath, Entry aliasedEntry) throws SystemDatabaseException;	指定されたエントリを参照する、指定されたパス名の新しいエイリアスを作成する。そのエイリアスのエントリを返す。
public void populateTree (TreePopulator populator) throws SystemDatabaseException;	指定されたポピュレーション・オブジェクトを使用してツリーをエントリで埋める。
public void printTree(boolean verbose);	記述的な方法でツリーを印刷する。
public void recursePrintTree (Cursor cursor, String prefix, boolean verbose);	指定された記述法でツリーを印刷する。

【0226】TreePopulatorインタフェース＊　＊ス

表1－3　TreePopulatorインタフェース

メソッド	機能
public int getRootEntry();	ルート・エントリへの整数参照を返す。
public String getEntryName (int entry) throws SystemDatabaseException;	指定されたエントリの名前を返す。
public int getParentEntry(int entry) throws SystemDatabaseException;	指定されたエントリの親エントリを返す。
public int getFirstChildEntry (int entry) throws SystemDatabaseException;	指定されたエントリの最初の子エントリを返す。
public int getPeerEntry(int entry) throws SystemDatabaseException;	指定されたエントリの次の兄弟エントリを返す。
public int getPropertyNameLength();	属性名の長さの最大値を返す。
public String getNextProperty(int entry, String prevPropName) throws SystemDatabaseException;	以前の属性名が与えられた次の属性名を返す。
public int getPropertyValueLength (int entry, String propName) throws SystemDatabaseException;	属性値の長さをバイト数で返す。

public Object getPropertyValue getPropertyValue(int entry, String propName) throws SystemDatabaseException;	属性値を返す。

【０２２７】Ｍａｎａｇｅｒインタフェース

表１－４　Ｍａｎａｇｅｒインタフェース

メソッド	機能
public Entry makeNewEntry(String name) throws SystemDatabaseException;	名前が与えられた新しいエントリ（ドラフト・エントリ）を作成する。
public String getPropertyPrintString (Entry e, String propName) throws SystemDatabaseException;	指定されたエントリの命名された属性のデコード／フォーマットされたＳｔｒｉｎｇ表現を獲得する。

【０２２８】クラス　＊　＊ＳｙｓｔｅｍＤａｔａｂａｓｅクラス

表１－５　ＳｙｓｔｅｍＤａｔａｂａｓｅクラス

メソッドまたはコンストラクタ	機能
public SystemDatabase(String[] namespaces, String[] mgrClasses) throws SystemDatabaseException;	システム・データベースを初期化する。最初の名前空間およびｍａｎａｇｅｒクラスを与える。
public static Entry getSuperRootEntry();	スーパー・ルートのエントリ参照を返す。
public static Tree getSystemDatabase();	システム・データベースを定義するツリーを返す。

【０２２９】ＳｙｓｔｅｍＥｎｔｒｙクラス　※ｓｔｅｎｔＳｙｓｔｅｍＥｎｔｒｙを設けることができる。

Ｅｎｔｒｙインタフェースを実装し、データベースの基本機能を提供する。機能を拡張するために、ＳｙｓｔｅｍＥｎｔｒｙクラスのサブクラス、すなわちＰｅｒｓｉ※

【０２３０】Ｔｒａｎｓａｃｔｉｏｎクラス

表１－６　Ｔｒａｎｓａｃｔｉｏｎクラス

メソッドまたはコンストラクタ	機能
public Transaction(Entry subtreeRoot)	指定されたｓｕｂｔｒｅｅＲｏｏｔエントリによって表された全てのエントリから成る新しい有効トランザクションを構成する。
final public void commit() throws SystemDatabaseException;	トランザクションをコミットする（全てのロックを解除する）。
final public void abort();	トランザクションを打ち切る（全てのロックをロールバックし解除する）。
final public boolean isValid();	トランザクションが有効か、または無効かを指示する。

【０２３１】Ｑｕｅｒｙクラス

表１－７　Ｑｕｅｒｙクラス

メソッドまたはコンストラクタ	機能
public Query(Entry entry, int scope);	指定された範囲の中の全てのエントリを突き合わせる、「ｅｎｔｒｙ」をルートとする問合せを作成する。
public Query(Entry entry, String name, int scope)	指定された範囲の中でコンポーネント名「ｎａｍｅ」を有するエントリを探索するための、「ｅｎｔｒｙ」をルートとする問合せを作成する。
public Query(Entry entry, String[] n	指定された範囲の中で、「ｎａｍｅｓ

77

ames, int scope)	」の中のいずれかと一致するコンポーネント名を有するエントリを探索するための、「ｅｎｔｒｙ」をルートとする問合せを作成する。
public int setSearchScope()	この探索の範囲を変更する。
public int getSearchScope()	この探索の範囲を取得する。
public String getSearchName()	この探索に使用されたエントリ名を返す。
public Object getSearchValue()	この探索に使用されたエントリ値を返す。
public boolean namesMustMatch()	名前は一致しなければならないか？
public Entry previousMatch()	最後の一致に問合せをリセットする。
public Entry getCurrentMatch()	現在の一致を返す。
public Entry getPreviousMatch()	最後の一致を返す。
public Entry nextMatch()	次の一致を返す。
public boolean hasMoreElements()	この探索で別の一致があるか？
public Object nextElement()	次の一致を返す。
final public static int THIS = 0;	探索範囲：自体を探索する。
final public static int PARENT =1;	探索範囲：親エントリを探索する。
final public static int SIBLINGS = 2;	探索範囲：エントリの全ての兄弟エントリを探索する。
final public static int CHILDREN = 3;	探索範囲：エントリの全ての直接の子エントリを探索する。
final public static int DESCENDANTS = 4;	探索範囲：自体を除くエントリの下の全てのエントリを探索する。
final public static int styleIsExact = 0;	突合せスタイル：名前が正確に一致しなければならない。
final public static int styleIsStartsWith =1;	突合せスタイル：名前の最初の文字が同じであればよい。

【０２３２】ＰｒｏｐｅｒｔｙＱｕｅｒｙクラス　30＊探索機能を提供する。
Ｑｕｅｒｙクラスを拡張し、エントリの属性名に基づく＊

表１－８　ＰｒｏｐｅｒｔｙＱｕｅｒｙクラス

メソッドまたはコンストラクタ	機能
public PropertyQuery(Entry entry, String name, int scope)	指定された範囲の中で属性名「ｎａｍｅ」を有するエントリを探索するための、「ｅｎｔｒｙ」をルートとする問合せを作成する。
public PropertyQuery(Entry entry, String[] names, int scope)	指定された範囲の中で「ｎａｍｅｓ」の中のいずれかと一致する属性名を有するエントリを探索するための、「ｅｎｔｒｙ」をルートとする問合せを作成する。

【０２３３】イベント・クラス　※メソッド　機能
ＳｙｓｔｅｍＤａｔａｂａｓｅＥｖｅｎｔクラス　なし。タグ・クラスのみ。
システム・データベースに関係した全てのイベントのベース・クラス。　【０２３４】ＥｎｔｒｙＥｖｅｎｔクラス
表１－９　ＳｙｓｔｅｍＤａｔａｂａｓｅＥｖｅｎｔクラス　※　ＳｙｓｔｅｍＤａｔａｂａｓｅＥｖｅｎｔを拡張し、全てのエントリ特定データベース・イベントのベースを表す。

表１－１０　ＥｎｔｒｙＥｖｅｎｔクラス

メソッド	機能

public Entry getEntry();	イベントが関係するエントリの参照を返す。

【0235】EntryInsertEventクラス　＊を指示する。
EntryEventを拡張する。公表エントリの挿入＊

表1－11　EntryInsertEventクラス

メソッド	機能
public Entry getEntry();	新しく挿入されたエントリの参照を返す。

【0236】EntryDisconnectEven　※EntryEventを拡張する。公表エントリの分離
tクラス　※10　を指示する。

表1－12　EntryDisconnectEventクラス

メソッド	機能
public Entry getEntry();	分離されたエントリへの参照を返す。
public Entry getAffectedParent();	前の親エントリへの参照を返す。ヌルの場合、エントリは分離点ではなく、分離点のエントリの子孫エントリである。

【0237】EntryRemoveEventクラス　★を指示する。
EntryEventを拡張する。公表エントリの削除★

表1－13　EntryRemoveEventクラス

メソッド	機能
public Entry getEntry();	削除されたエントリへの参照を返す。
public Entry getAffectedParent();	前の親エントリへの参照を返す。

【0238】EntryPropertyEventク　☆エントリ属性特定データベース・イベントのベースを表
ラス　す。
EntryEventを拡張する。挿入を表し、全ての☆

表1－14　EntryPropertyEventクラス

メソッド	機能
public Entry getEntry();	属性イベントが関係するエントリの参照を返す。
public String getPropertyName();	影響を受けた属性の名前を返す。

【0239】EntryPropertyInsert　◆EntryPropertyEventを拡張する。公
Eventクラス　◆　表エントリに新しい属性が追加されたことを指示する。

表1－15　EntryPropertyInsertEventクラス

メソッド	機能
public Entry getEntry();	新しい属性が追加されたエントリへの参照を返す。
public String getPropertyName();	新しい属性の名前を返す。

【0240】EntryPropertyRemove　＊EntryPropertyEventを拡張する。公
Eventクラス　＊40　表エントリから属性が削除されたことを指示する。

表1－16　EntryPropertyRemoveEventクラス

メソッド	機能
public Entry getEntry();	属性が削除されたエントリの参照を返す。
public String getPropertyName();	影響を受けた属性の名前を返す。
public Object getPreviousValue();	削除された属性の以前の値を返す。

【0241】EntryPropertyValueC　※EntryPropertyEventを拡張する。公
hangeEventクラス　※　表エントリの属性値が変更されたことを指示する。

表1－17　EntryPropertyValueChangeEventク
ラス

メソッド　機能
public Entry getEntry();　属性値が変更されたエントリの参照を返す。
public String getPropertyName();　影響を受けた属性の名前を返す。
public Object getPreviousValue();　属性の以前の値を返す。

【0242】例外クラス
SystemDatabaseExceptionクラス
システム・データベースに関係した全ての例外クラスのベース・クラス。
表1－18　SystemDatabaseExceptionクラス

＊メソッド　機能
なし。タグ・クラスのみ。

【0243】EntryExceptionクラス
SystemDatabaseExceptionを拡張し、全てのエントリ特定データベース例外クラスのベース・クラスを表す。
表1－19　EntryExceptionクラス

メソッド　機能
public Entry getEntry();　イベントが関係するエントリの参照を返す。

【0244】EntryInvalidStateExceptionクラス
EntryExceptionを拡張する。エントリが※期待される状態にないことを指示する。例えば、公表エントリを挿入しようとすると、この例外が送出される。
表1－20　EntryInvalidStateExceptionクラス

メソッド　機能
public Entry getEntry();　イベントが関係するエントリの参照を返す。

【0245】EntryLockExceptionクラス
EntryExceptionを拡張し、ロックに関係★した全てのエントリ特定データベース例外クラスのベース・クラスを表す。
表1－21　EntryLockExceptionクラス

メソッド　機能
public Entry getEntry();　イベントが関係するエントリの参照を返す。
SystemDatabaseLock getLock();　イベントが関係するロックへの参照を返す。

【0246】EntryLockStateExceptionクラス
EntryLockExceptionを拡張する。エ☆ントリ・ロックが期待される状態にないことを指示する。
表1－22　EntryLockStateExceptionクラス

メソッド　機能
public Entry getEntry();　イベントが関係するエントリの参照を返す。
SystemDatabaseLock getLock();　イベントが関係するロックへの参照を返す。

【0247】EntryTimeoutExceptionクラス
EntryLockExceptionを拡張する。エ◆ントリ・ロック操作がタイムアウトしたことを指示する(デッドロックの検出)。
表1－23　EntryTimeoutExceptionクラス

メソッド　機能
public Entry getEntry();　イベントが関係するエントリの参照を返す。
SystemDatabaseLock getLock();　イベントが関係するロックへの参照を返す。

【0248】EntryNameExistsExceptionクラス
EntryExceptionを拡張する。そのエント＊リの名前が、特定の親エントリの下にすでに存在することを指示する。

表1－24 EntryNameExistsExceptionクラス

メソッド	機能
public Entry getEntry();	同一の名前をすでに有するエントリの参照を返す。

【0249】EntryNotFoundExceptionクラス
※EntryExceptionを拡張する。そのエントリ名が存在しないことを指示する。※

表1－25 EntryNotFoundExceptionクラス

メソッド	機能
public Entry getEntry();	パス名のルックアップが最後に成功したエントリの参照を返す。

【0250】EntryPropertyNameNotFoundExceptionクラス
★EntryExceptionを拡張する。そのエントリ属性名が存在しないことを指示する。★

表1－26 EntryPropertyNameNotFoundExceptionクラス

メソッド	機能
public Entry getEntry();	この例外が関係するエントリの参照を返す。
public String getName();	見つからなかった名前を返す。

【0251】EntryPropertyNameInvalidExceptionクラス
EntryExceptionを拡張する。エントリ属☆性名が、エントリ・クラスに対して無効であることを示す。☆

表1－27 EntryPropertyNameInvalidExceptionクラス

メソッド	機能
public Entry getEntry();	この例外が関係するエントリの参照を返す。
public String getName();	無効であった名前を返す。

【0252】TransactionInvalidExceptionクラス
SystemDatabaseExceptionを拡張する。トランザクションに打切りが実行されたため◆に、トランザクション・オブジェクトが無効となり、それ以降、そのオブジェクトを使用することができなくなったことを指示する。◆

表1－28 TransactionInvalidExceptionクラス

メソッド	機能
public Transaction getTransaction();	イベントが関係するトランザクションの参照を返す。

【図面の簡単な説明】

【図1】 本発明を実施したコンピュータ・システムのハードウェア・ブロック図である。

【図2】 図1のコンピュータ・システムのオブジェクト指向ソフトウェアを示す図である。

【図3】 割込み源間の関係の例を示す図である。

【図4】 図3の例に対応する割込み源ツリーの一部分を示す図である。

【図5】 バス割込み源エントリの例を示す図である。

【図6】 割込み名前空間とデバイス名前空間の相互参照を示す図である。

【図7】 割込み源ツリーのさまざまなレベルとソフトウェアとの関係を示す図である。

【図8】 割込みハンドラの同期を示す図である。

【図9】 据置き割込み処理の概念を示す図である。

【図10】 割込みディスパッチャ、および割込みディスパッチャと実行時ソフトウェア、マイクロカーネル・ソフトウェアとの関係を示す図である。

【図11】 バス割込みハンドラおよびデバイス割込みハンドラを示す図である。

【図12】 割込み源クラス階層を示す図である。

【図13】 本発明による、イーサネット読取り割込み動作の性能向上を示す図である。

【図14】 JSDの構成を示す図である。

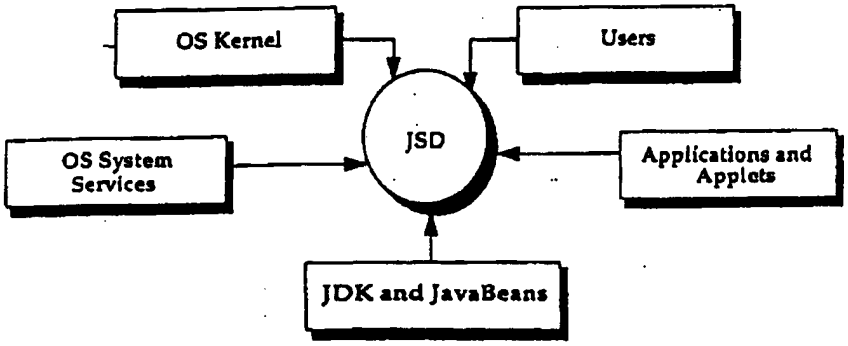
85

【図１５】　ＪＳＤのポピュレーションと使用方法を示す図である。
【図１６】　標準名前空間の最上位レベルを示す図である。
【図１７】　ｓｏｆｕｔｏｗａｒｅ名前空間のサブスキーマを示す図である。
【図１８】　ａｐｐｌｉｃａｔｉｏｎ階層を示す図である。
【図１９】　ｓｙｔｅｍ階層を示す図である。
【図２０】　Ｓｅｒｖｉｃｅ階層を示す図である。
【図２１】　Ｐｕｂｌｉｃ階層を示す図である。
【図２２】　バスーデバイス・トポロジを示す図である。
【図２３】　インターフェース・トポロジを示す図である。
【図２４】　Ｉｎｔｅｒｆａｃｅ名前空間の割り当てを示す図である。
【図２５】　エントリの状態遷移を示す図である。
【図２６】　サブツリーの挿入を示す図である。
【図２７】　サブツリーの分離を示す図である。
【図２８】　サブツリーの除去の最終段階を示す図である。
【図２９】　ＪＳＤ絵弁とのクラス階層を示す図である。
【図３０】　サブツリーの定義を示す図である。
【図３１】　相互作用を示す図である。
【図３２】　トランザクションとロック例外を示す図である。
【図３３】　分散トランザクション・モデルを示す図である。
【図３４】　ＪＳＤ例外階層を示す図である。
【図３５】　ナビゲーションの例を示す図である。
【図３６】　ナビゲーションの例を示す図である。
【図３７】　ｃｏｎｆｉｇ名前空間のサブスキーマを示す図である。

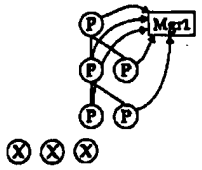
86

＊【図３８】　マシン階層を示す図である。
【図３９】　ユーザ階層を示す図である。
【図４０】　ＪＳＤのクライアント／サーバ編成を示す図である。
【符号の説明】
３２　プラットフォーム・インデペンデント層
３４　プラットフォーム・デペンデント
３６　アプリケーション・プログラム層
３８　実行時システム
４０　Ｊａｖａ仮想マシン
４２　デバイス・インタフェース
４４　バス・マネージャ
４６　デバイスマネージャ
４８　各種マネージャ
５０　デバイス・ドライバ
５１　プラットフォーム・インデペンデント・バス・マネージャ
５２　プラットフォーム・インデペンデント・メモリ
５４　システム・ローダ
５６　システム・データベース
５８　追加機能
６０　プラットフォーム・インタフェース
６１　ＯＳネイティブ層
６２　マイクロカーネル
６４　ブート・インタフェース
６６　仮想マシン・システム機能ライブラリ・ハンドラ
６８　割込みクラス
７０　ダイレクト・メモリ・アクセス（ＤＭＡ）クラス
７２　メモリ・クラス
７４　カーネル機能
７６　デバッグ機能
７８　割込みネイティブ・メソッド
８０　ＤＭＡネイティブ・メソッド
８２　メモリ・ネイティブ・メソッド

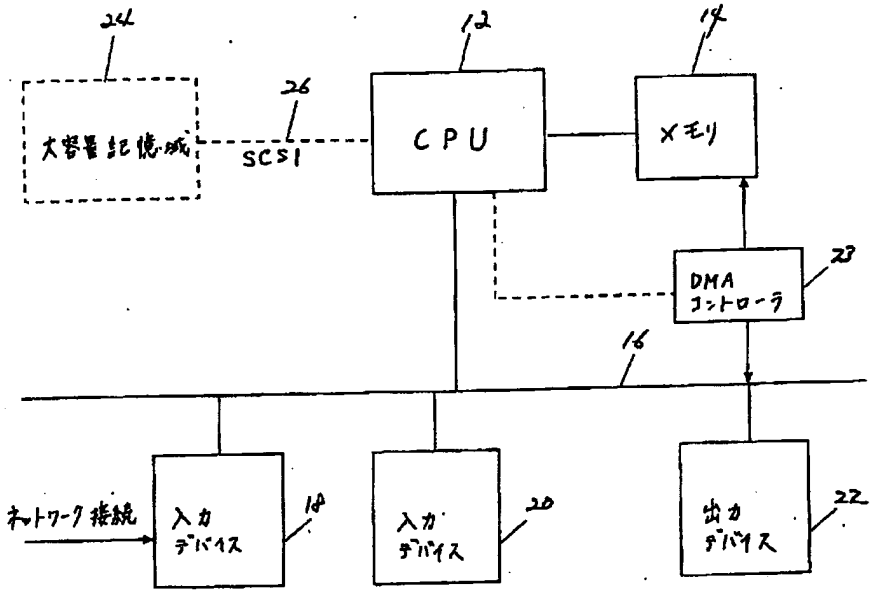
【図１４】



【図２８】

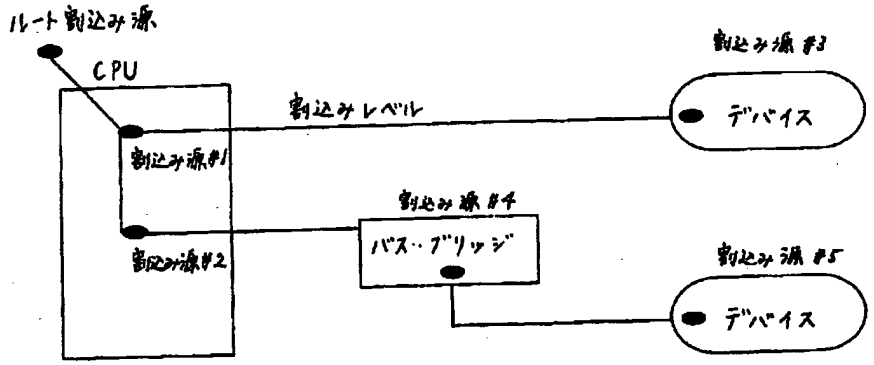


【図1】

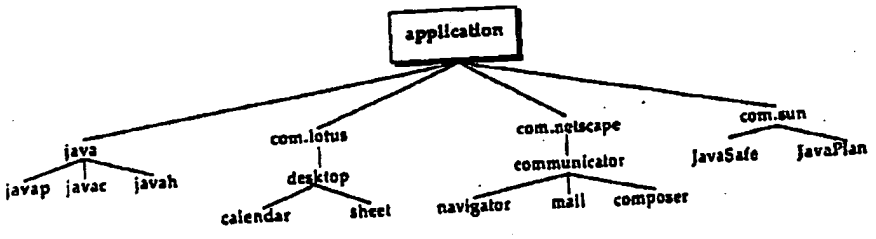


【図3】

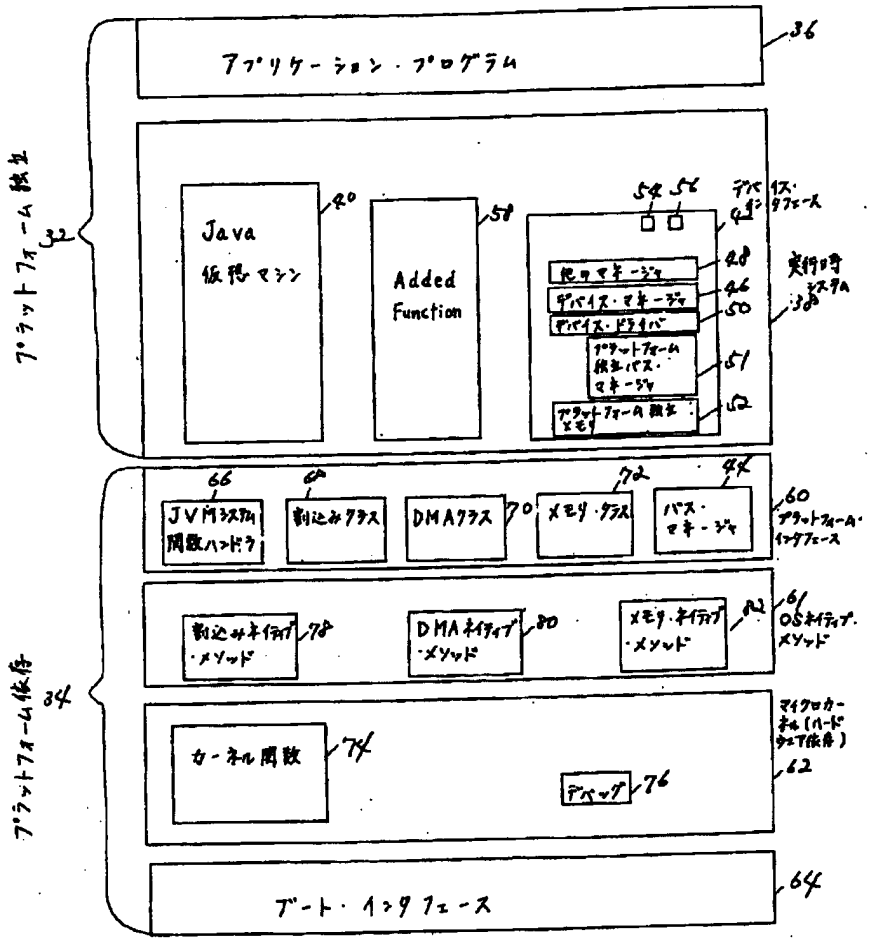
割込み源の関係



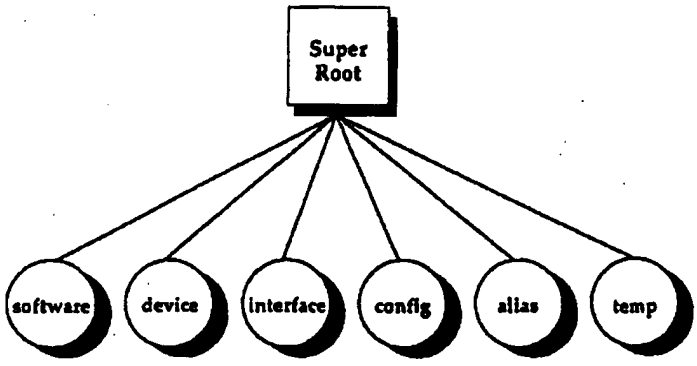
【図18】



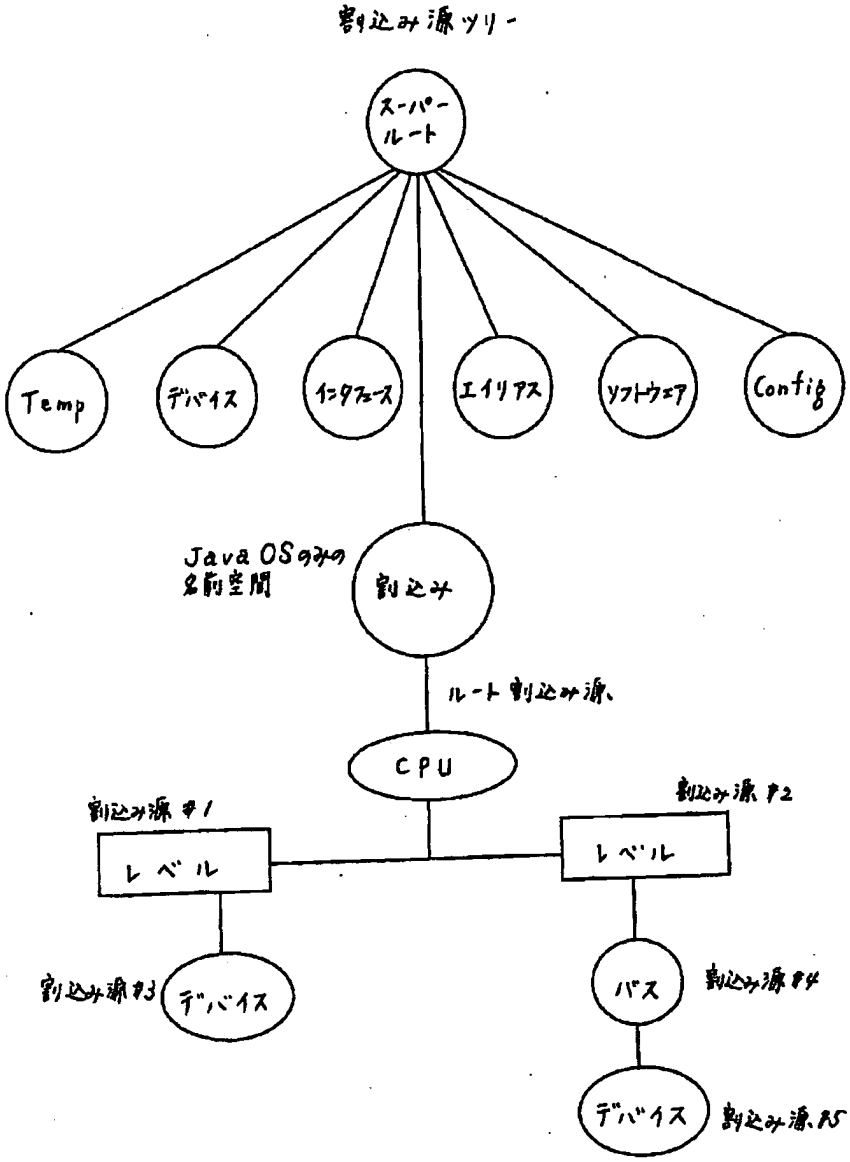
【図2】


アプリケーション・プログラム
36
プラットフォーム独立
32
Java
仮想マシン
40
Added
Function
58
54
56
デバイス・インタフェース
42
他のマネージャ
48
デバイス・マネージャ
46
デバイス・ドライバ
50
プラットフォーム独立バス・マネージャ
51
プラットフォーム独立メモリ
52
実行時システム
38
JVMシステム関数ハンドラ
66
割込みクラス
68
DMAクラス
70
メモリ・クラス
72
バス・マネージャ
74
60
プラットフォーム・インタフェース
割込みネイティブ・メソッド
78
DMAネイティブ・メソッド
80
メモリ・ネイティブ・メソッド
82
61
OSネイティブ・メソッド
プラットフォーム依存
34
カーネル関数
74
デバッグ
76
マイクロカーネル（ハードウェア依存）
62
ブート・インタフェース
64

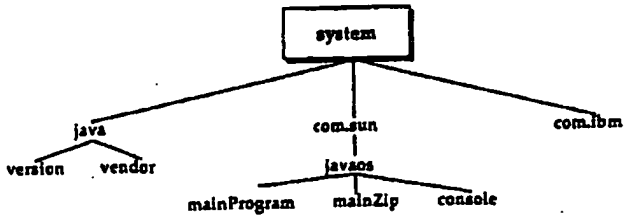
【図16】


Super Root
software
device
interface
config
alias
temp

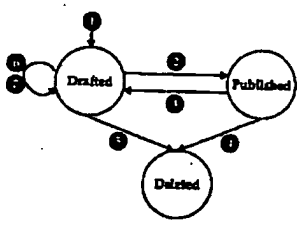
【図4】


割込み源ツリー
スーパー
ルート
Temp
デバイス
インタフェース
エイリアス
ソフトウェア
Config
Java OSのみの
名前空間
割込み
ルート割込み源、
CPU
割込み源 #1
レベル
割込み源 #2
レベル
割込み源 #3
デバイス
バス
割込み源 #4
デバイス
割込み源 #5

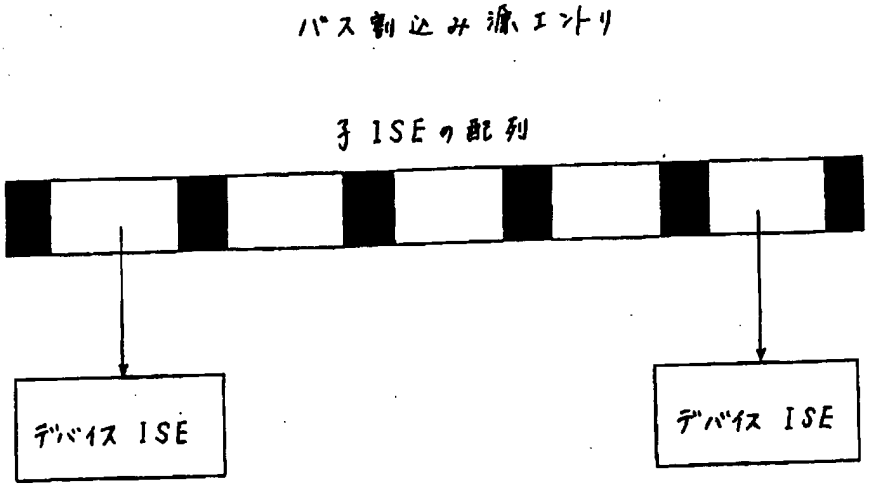
【図19】


system
java
version
vendor
com.sun
javaos
mainProgram
mainZip
console
com.ibm

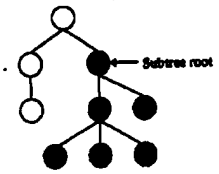
【図25】


Drafted
Published
Deleted

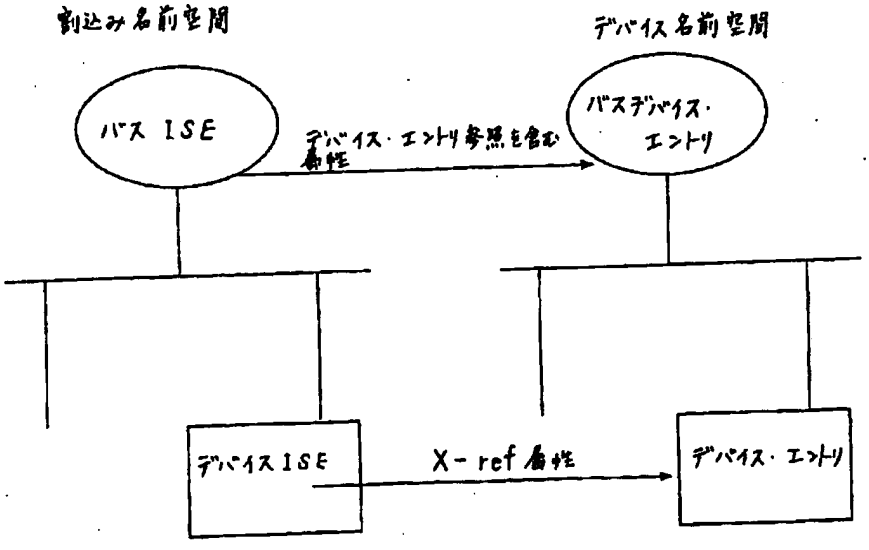
【図5】


バス割込み源エントリ
子ISEの配列
デバイス ISE
デバイス ISE

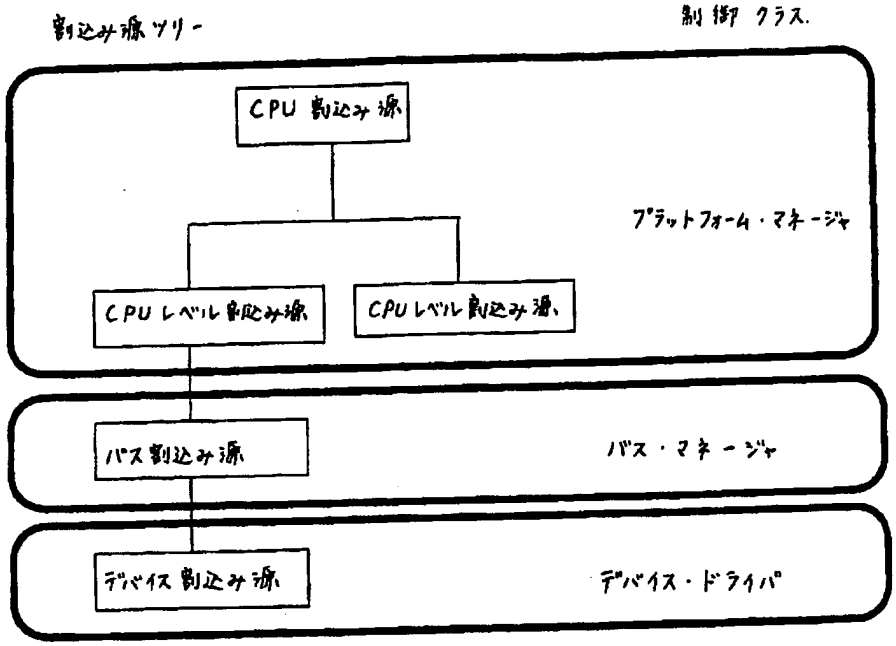
【図30】


Subtree root

【図6】

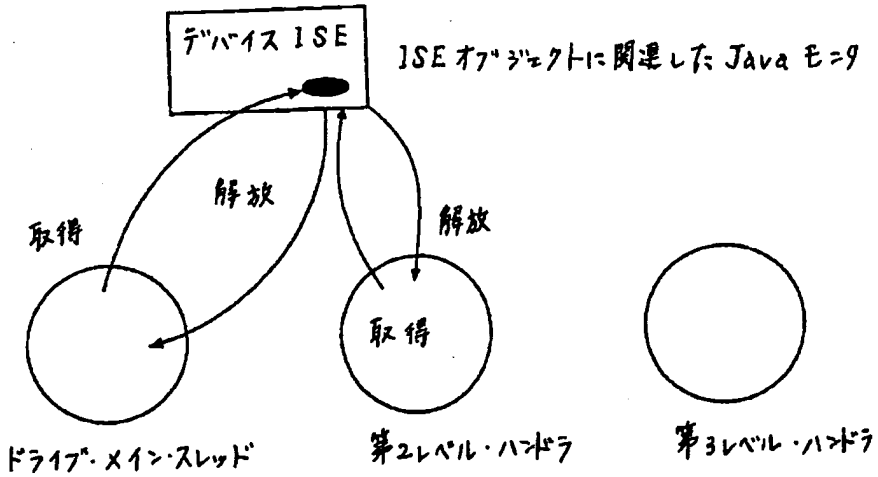

割込み名前空間
デバイス名前空間
バス ISE
デバイス・エントリ参照を含む属性
バスデバイス・エントリ
デバイスISE
X-ref属性
デバイス・エントリ

【図7】

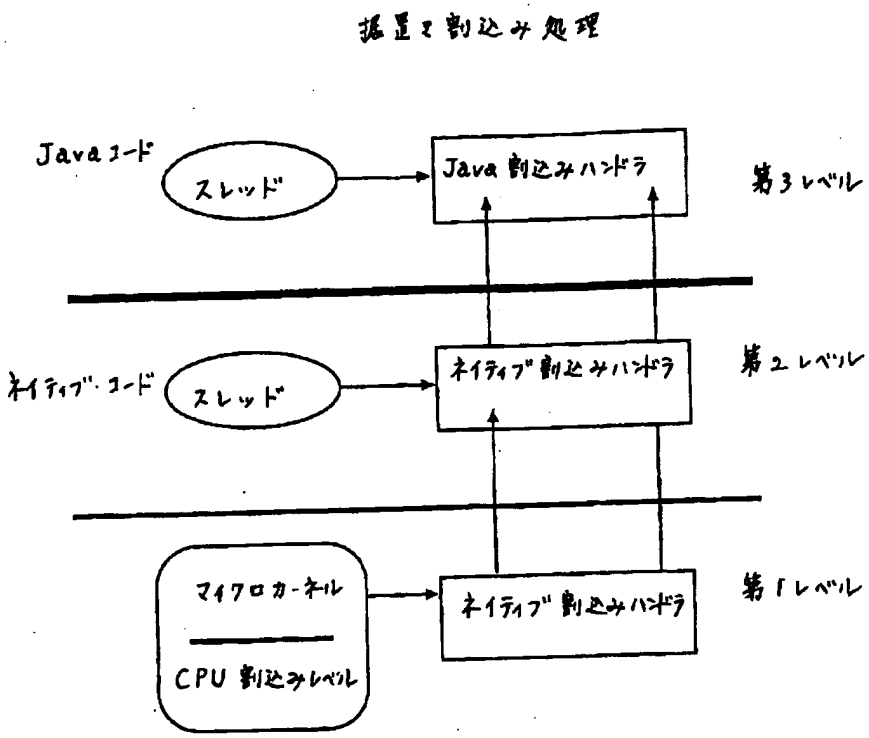


【図8】

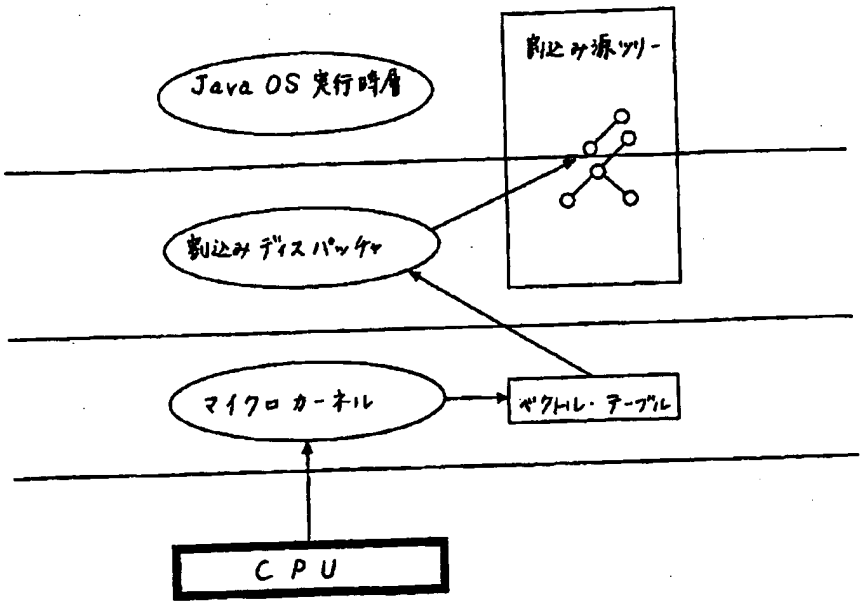
割込みハンドラの同期



【図9】

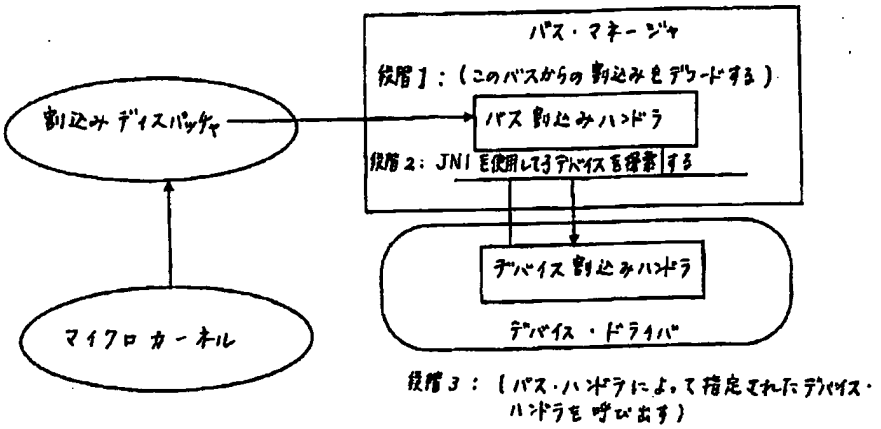

Javaコード
スレッド
Java割込みハンドラ
第3レベル
ネイティブ・コード
スレッド
ネイティブ割込みハンドラ
第2レベル
マイクロカーネル
CPU割込みレベル
ネイティブ割込みハンドラ
第1レベル

【図10】


Java OS実行時層
割込み源ツリー
割込みディスパッチャ
マイクロカーネル
ベクトル・テーブル
CPU

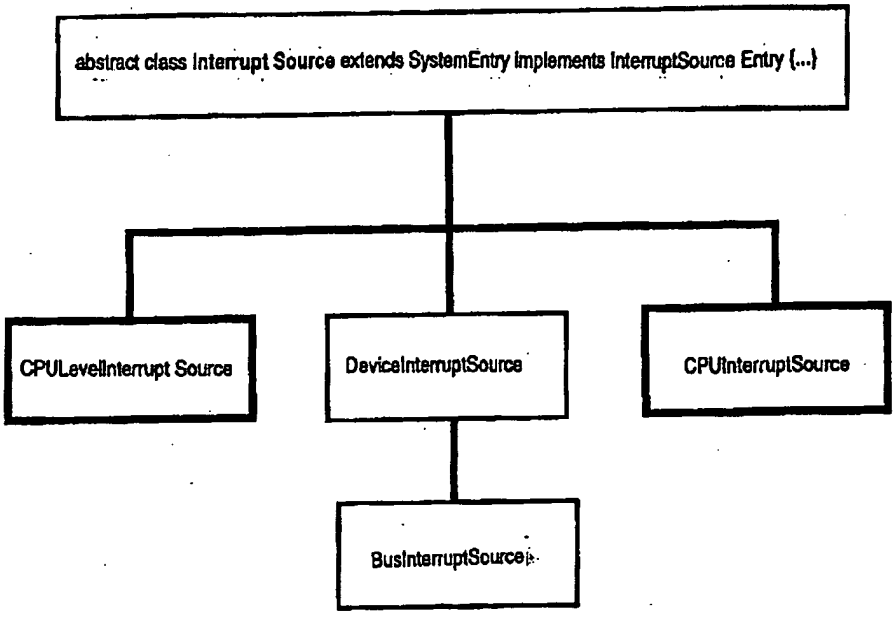
【図１１】

バス および デバイス 割込みハンドラ



【図１２】

割込み源クラス 階層



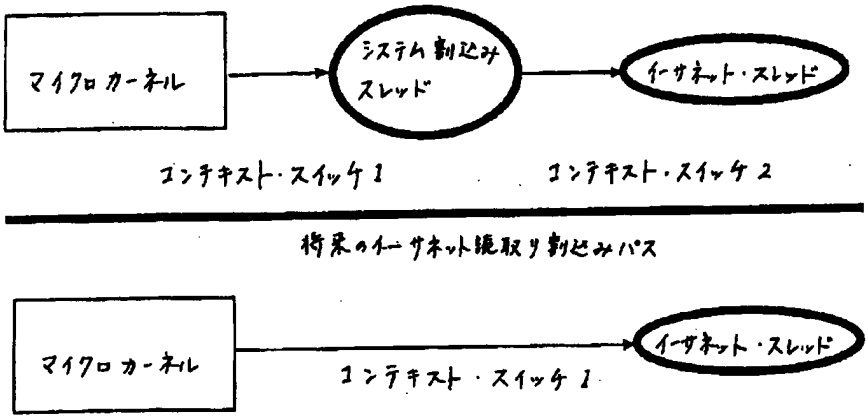
凡例

抽象クラス

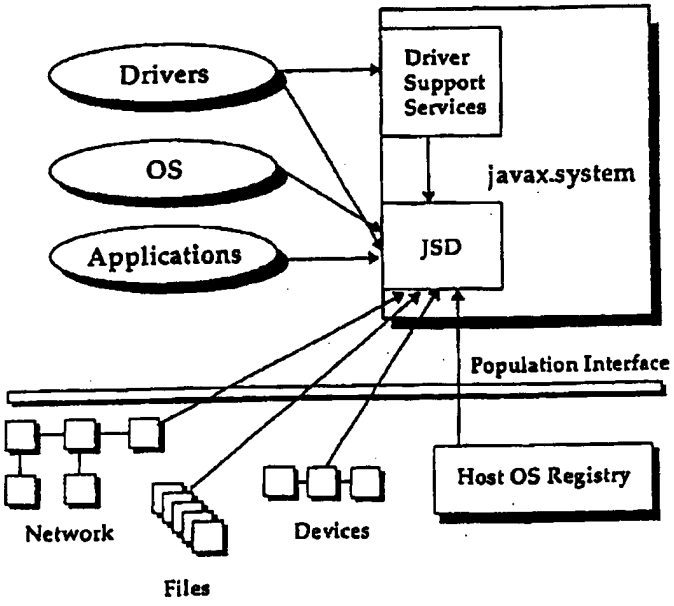
具象クラス

【図１３】

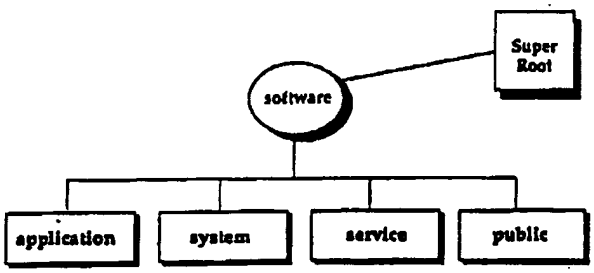
現在のイーサネット読取り割込みパス



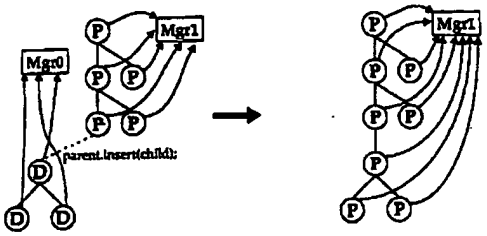
【図１５】



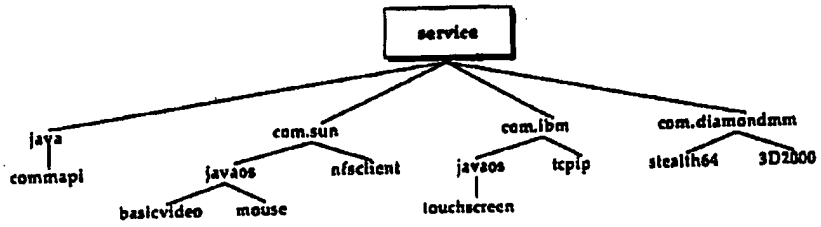
【図17】



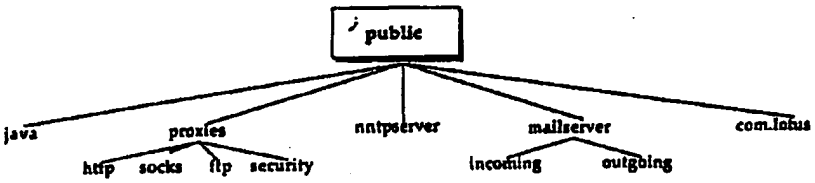
【図26】



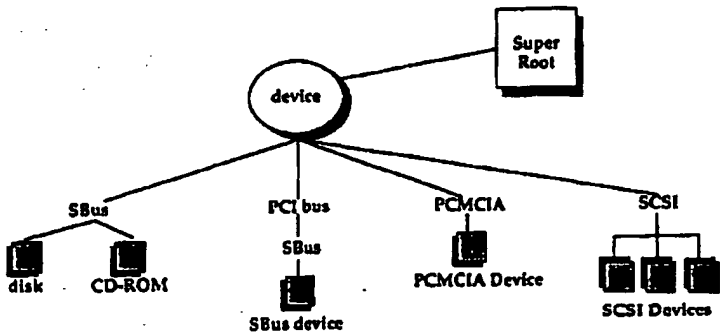
【図20】



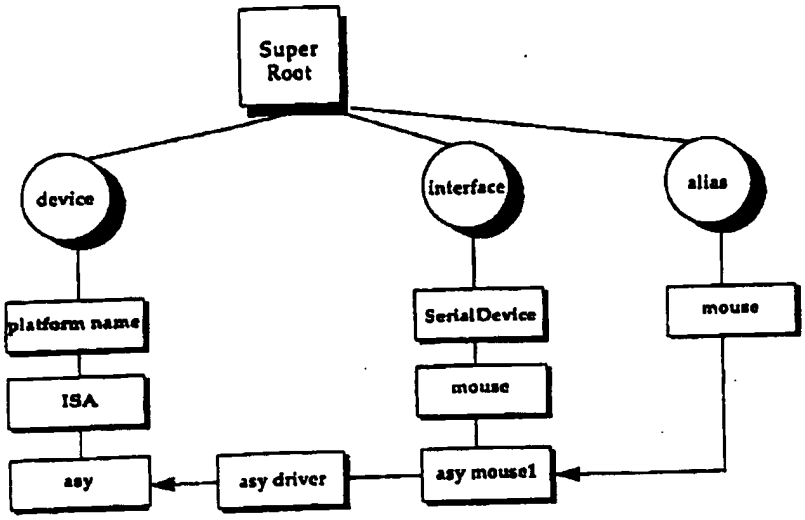
【図21】



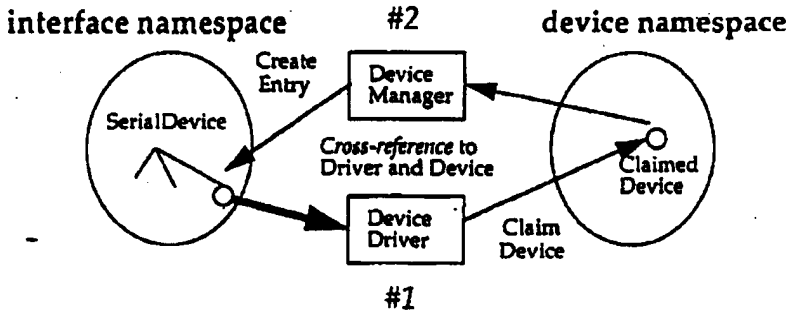
【図22】



【図23】

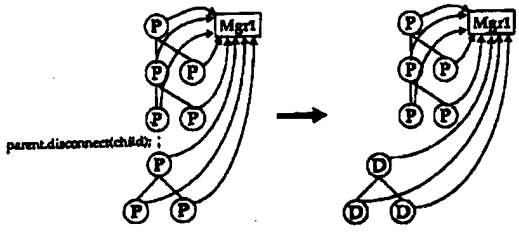


【図24】

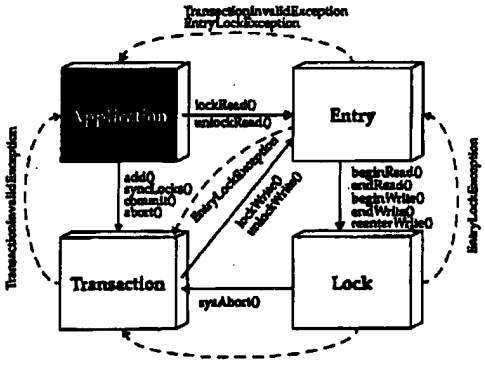



```
class DeviceDriver implements SerialDevice {
.....
}
```

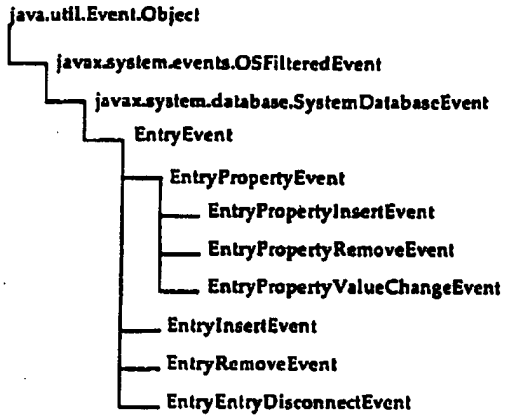
【図27】



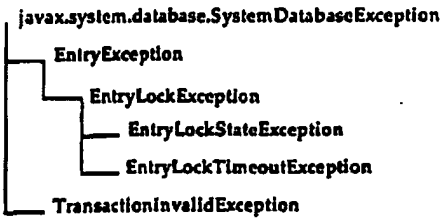
【図31】



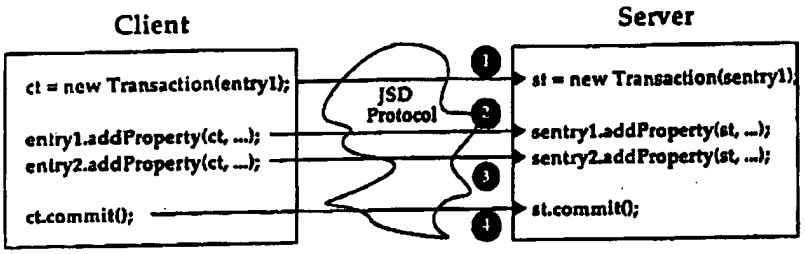
【図２９】



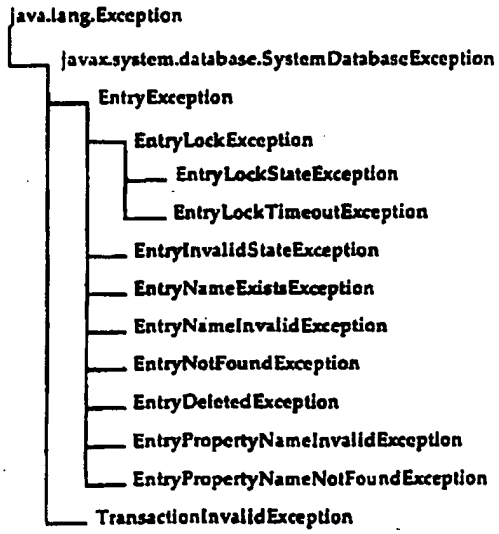
【図３２】



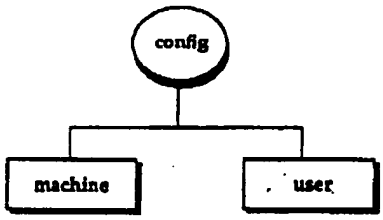
【図３３】



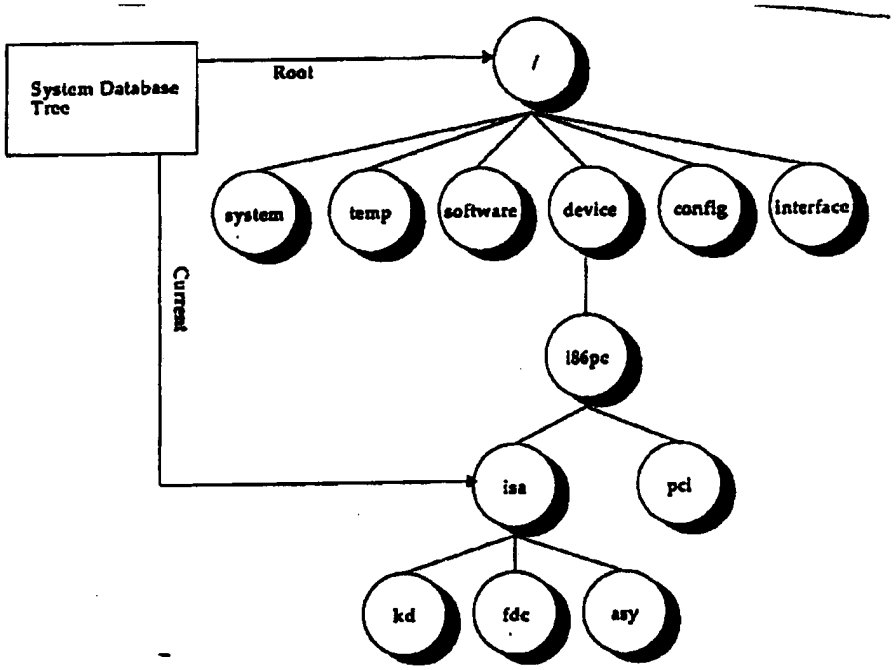
【図３４】



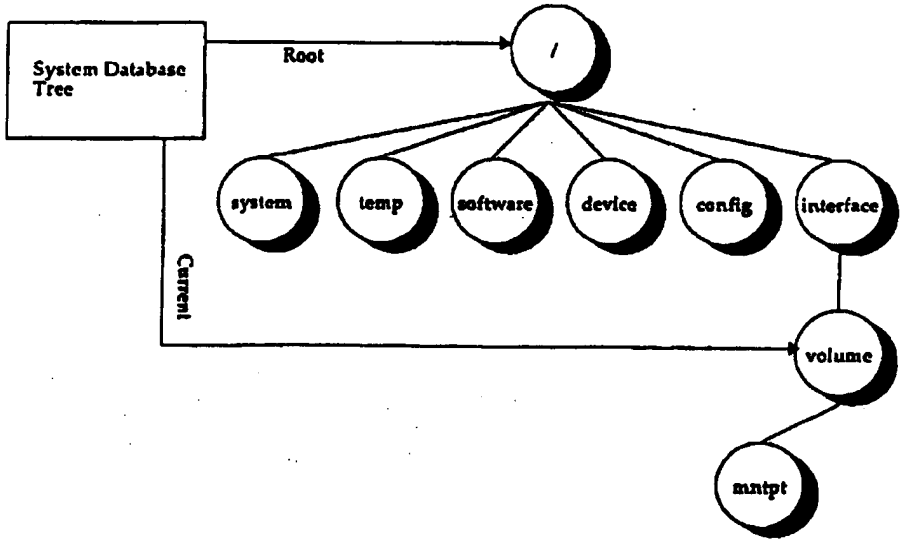
【図３７】



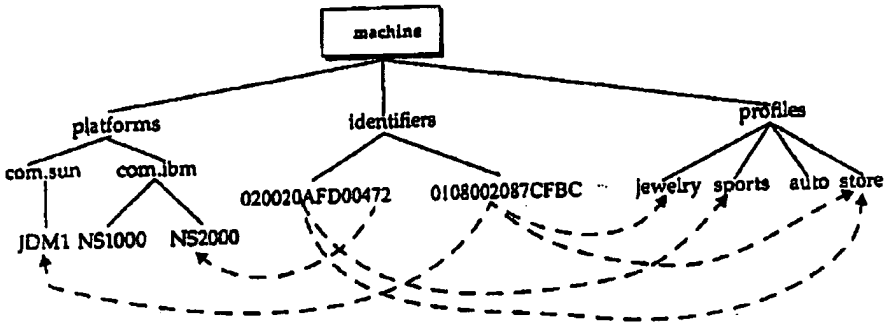
【図３５】


System Database Tree
Root
/
system
temp
software
device
config
interface
Current
i86pc
isa
pci
kd
fdc
asy

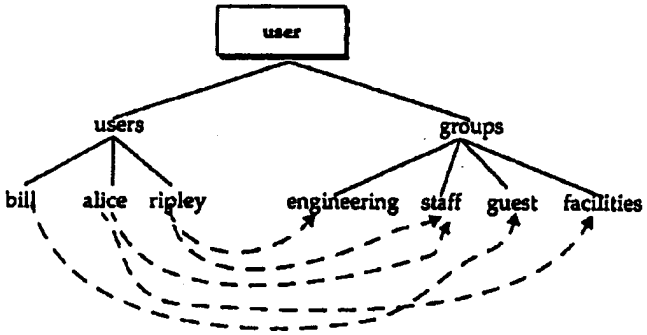
【図３６】


System Database Tree
Root
/
system
temp
software
device
config
interface
Current
volume
mntpt

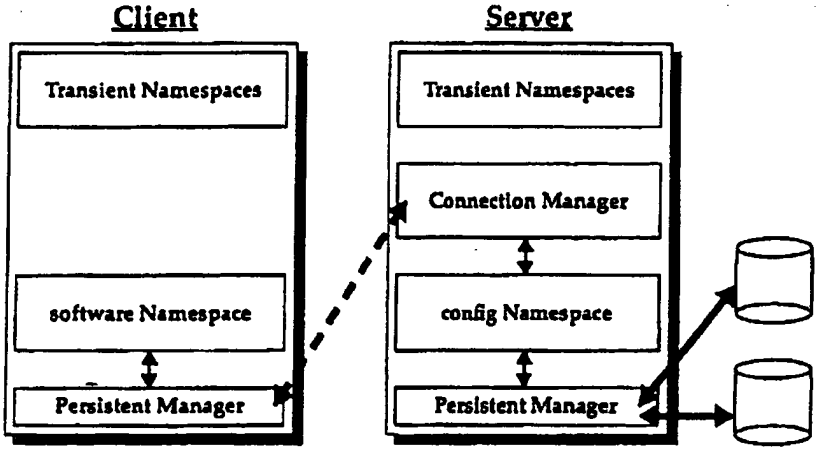
【図３８】


machine
platforms
identifiers
profiles
com.sun
com.ibm
020020AFD00472
0108002087CFBC
jewelry
sports
auto
store
JDM1
NS1000
NS2000

【図３９】


user
users
groups
bill
alice
ripley
engineering
staff
guest
facilities

【図４０】


Client
Server
Transient Namespaces
Transient Namespaces
Connection Manager
software Namespace
config Namespace
Persistent Manager
Persistent Manager

フロントページの続き

(71)出願人 591064003
901 SAN ANTONIO ROAD
PALO ALTO, CA 94303, U. S. A.
(72)発明者 トーマス・サウルバウグ
アメリカ合衆国・95120・カリフォルニア州・サン ホゼ・ブレート ハート ドライブ・6938
(72)発明者 グレゴリー・ケイ・スローター
アメリカ合衆国・94306・カリフォルニア州・パロ アルト・エマーソン ストリート・3326
(72)発明者 シャオヤン・ツェン
アメリカ合衆国・94555・カリフォルニア州・フレモント・ゴルビン コモン・5454